Departamento de ingeniería Telemática

PROYECTO FIN DE CARRERA

Ingeniería de telecomunicación

Monitorización del consumo eléctrico con sensores Waspmote

Autor: Fernando Moreno Martín
Directora: María Calderón Pastor

Leganés, Septiembre 2015

"El triunfo no está en vencer siempre, sino en nunca desanimarse. Es el triunfo el que crea al gran hombre"

Napoleón Bonaparte

Gracias a toda mi familia y amigos por estar ahí.
Dream chaser

Resumen

El proyecto consiste en la monitorización del consumo eléctrico de los electrodomésticos de un hogar mediante dispositivos de bajo coste.

A este fin se ha seleccionado una plataforma modular open source para construir redes de sensores inalámbricas: Waspmote.

Para ello se ha desarrollado una aplicación para esta plataforma que permite tomar muestras de corriente por medio de un sensor y trasmitirlas mediante una red WAN a un ordenador en donde se almacenarán.

Así mismo se ha implementado una aplicación cliente-servidor que solicita, recibe y almacena en una base de datos local, las muestras de corriente tomadas en el dispositivo Waspmote, presentando la información de una forma muy clara, permitiendo tomar una serie de decisiones relativas al consumo eléctrico.

Adicionalmente se ha desarrollado una aplicación Android para dispositivos móviles (smartphones y tablets) que permite solicitar y presentar en pantalla, de forma remota mediante una conexión WiFi, las muestras del consumo eléctrico de los diferentes dispositivos conectados a las sondas asociadas a la aplicación cliente-servidor.

Palabras clave: Monitorización, Waspmote, sensor de corriente, cliente-servidor, base de datos, WiFi, Android.

Abstract

The goal of this project consists of monitoring the power consumption of home appliances using low cost devices.

To achieve that, an open source platform has been chosen for building networks of wireless sensors: Waspmote.

For this application, it has been developed a platform that allows taking samples of current through a sensor and transmits them to a computer via WAN where they were stored.

It has also been implemented a client-server application that requests, receives and stores in a local database, the current samples taken in the Waspmote, presenting the information in a very clear way and allowing to take decisions related to energy consumption.

Additionally an Android application has been created for mobile devices (smartphones & tablets) that allows to request and displays remotely via WiFi, the sample of consumption of different devices connected to the probes associated to the client-server application.

Keywords: Monitoring, Waspmote, current sensor, client-server, database, WiFi, Android.

Índice

Lista de figuras

Índice de tablas

Capítulo 1

Introducción

En este primer capítulo se ofrecerá una introducción inicial sobre los objetivos y las distintas secciones con las que consta el presente proyecto fin de carrera.

1.1 Introducción

La energía eléctrica se ha convertido en un factor fundamental en el mundo moderno. Los electrodomésticos consumen energía eléctrica, sin embargo, pocas veces se han realizado estudios sobre lo que cuesta a los consumidores la energía de estos electrodomésticos y como contribuyen al gasto de la economía en el hogar.

Según los datos de la OCU en España hay 17 millones de viviendas habitadas y éstas consumen la quinta parte de toda la energía que se consume en España (y la cuarta parte de la electricidad).

Una casa española gasta de media 990 euros al año en energía distribuyéndose en nuestros hogares de la siguiente manera:

- Calefacción: media anual de 5.172 kWh
- Electrodomésticos: 1.924 kWh
- Agua caliente: 1.877 kWh
- Cocina: 737 kWh
- Iluminación: 410 kWh
- Aire acondicionado: 170 kWh

Siendo los electrodomésticos que más consumen en los hogares los siguientes:

- Frigorífico: 662 kWh al año
- Congelador: 563 kWh
- Televisión: 263 kWh.
- Lavadora: 255 kWh
- Secadora: 255 kWh.
- Lavavajillas: 246 kWh

Muchos usuarios ignoran el gasto que les supone tener aparatos que no están encendidos pero siguen enchufados (modo stand-by) como por ejemplo una televisión que no emite imágenes pero sigue con su piloto rojo iluminado, o el reloj parpadeante de un despertador, etc… Estos gastos que no se tienen tanto en cuenta suponen una media de 231 kWh al año por cada casa, es decir un 2,2% de todo nuestro consumo. Para hacerse a la idea dicho consumo es el mismo que el que un horno o un ordenador pueda consumir en un año

Por tanto la monitorización objeto de este proyecto, podría ayudar a reducir el consumo de electricidad, en su hogar o trabajo, reduciendo la factura, simplemente por adecuar el funcionamiento de los dispositivos eléctricos a la hora más adecuada en función de la tarifa eléctrica.

1.2. Objetivos

La finalidad de este proyecto es la diseñar un sistema que sea capaz de monitorizar de una manera sencilla y eficaz el consumo de los distintos dispositivos eléctricos que pueda haber en un hogar mediante la adquisición de muestras de corriente por medio de sensores Waspmotes.

El sistema debe ser capaz de tomar muestras de corriente de cualquier dispositivo eléctrico, para a continuación enviarlas a un servidor, en donde quedarán almacenadas en una base de datos, y poder llevar a cabo un control del consumo.

Por último se ha creado una aplicación en Android desde la cual un usuario puede solicitar el conocer el consumo instantáneo en amperios de un determinado electrodoméstico mediante una conexión a Internet.

A continuación se muestra una imagen que ilustra el escenario propuesto.

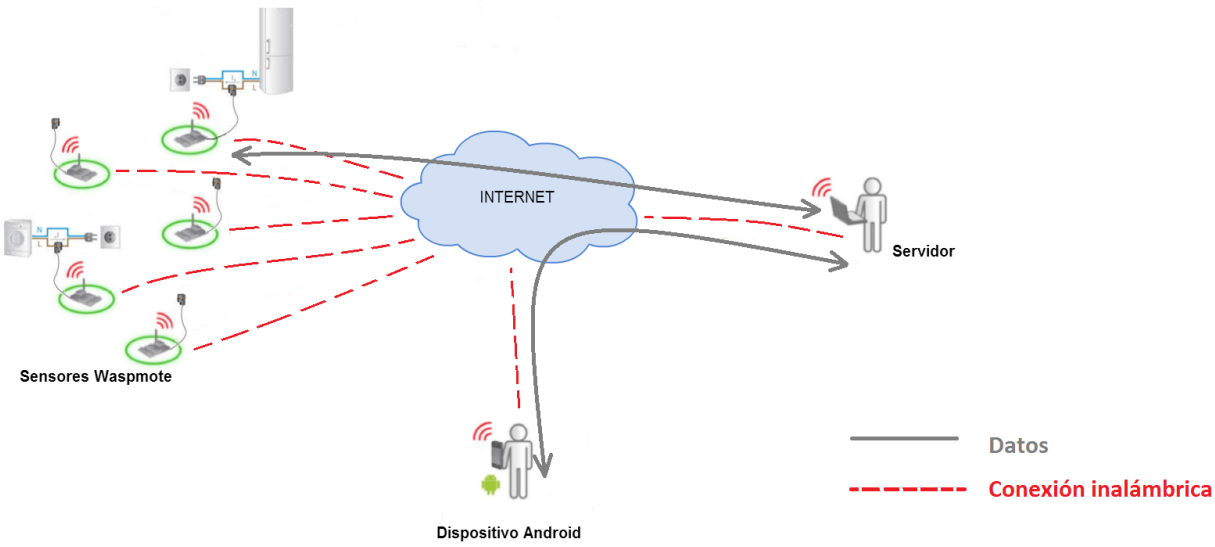

Figura 1: Solución propuesta del proyecto

1.3. Estructura del proyecto

La memoria del proyecto está constituida por cincos capítulos y anexos. A continuación se describe brevemente cada parte de la memoria.

- **Capítulo 2:** Estado del arte. Se muestran diversos estudios llevados a cabo para ahorrar energía. Además se muestran distintas tecnologías con las que se podría dar sustento a este proyecto. Y por último ejemplos de sensores de corriente que están empezando a ser desplegados en el mercado.

- **Capítulo 3:** Diseño de la solución propuesta. Se describen detalladamente los componentes utilizados para la realización del proyecto y las soluciones que se han implementado.

- **Capítulo 4:** Evaluación de la solución. Se describen las pruebas realizadas para comprobar el funcionamiento del código implementado.

- **Capítulo 5:** Conclusiones y líneas futuras. Se analiza si se han cumplido los objetivos del proyecto además de valorar los resultados obtenidos. También se comenta los trabajos futuros que se podrían realizar.

- En los **anexos** se incluyen los manuales de usuario, los manuales de instalación de los programas, la planificación del proyecto tanto en forma de tareas como temporal y el presupuesto del proyecto.

Capítulo 2

Estado del arte

En este capítulo se pretende dar una visión de las soluciones actuales para encontrar soluciones al ahorro del consumo energético de equipos eléctricos y electrónicos de los hogares.

También se enunciarán diferentes formas de transmitir la información recopilada por los sensores al servidor.

Por último se realizará una presentación de diversos sensores que están a la venta a día de hoy en el mercado.

2.1. Mecanismos de ahorro de energía

Los mecanismos para el ahorro de energía son diversos. A continuación se van a citar una serie de técnicas que han sido llevadas a cabo en investigaciones para reducir el consumo en los aparatos eléctricos especialmente en el sector doméstico.

La primera investigación utiliza una política de ahorro de energía adaptativa (AESP, adaptive energy saving policy). Para ello utiliza prioridades en el uso de los aparatos en función de la energía que consumen, pudiéndose reducir hasta en un 13% la energía consumida en un hogar [1]. La investigación, lleva a cabo un experimento con el que se comprobó que a diferentes horas del día se utilizaban más unos dispositivos que otros. Por ejemplo, el frigorífico es un electrodoméstico que está consumiendo las 24 horas del día. Las luces, los televisores y lavadoras funcionan regularmente durante cierto tiempo, mientras que los PCs y los aires

acondicionados o calefacciones funcionan irregularmente en base a la decisión del usuario. Y además, no a todos los dispositivos les afecta igual un apagón, por ejemplo, a los ordenadores les llevaría a una pérdida de información, imposible de recuperar si no se hubiera almacenado previamente.

Puesto en antecedentes, se concluyó que cuando la demanda de energía fuera inferior a la disponible, se suministraría energía a todos los aparatos del hogar, incluso a aquellos electrodomésticos con baja prioridad o con ninguna. Sin embargo, cuando la demanda de energía excediera a la de la fuente de alimentación, se suministraría energía a los equipos de acuerdo con la prioridad de uso establecida.

La energía es suministrada gradualmente a los aparatos de mayor prioridad, siendo bloqueada a aquellos dispositivos que excedan la energía disponible y tengan una prioridad menor. Además se creó un sistema de comunicación UCFEMS (Sistema de gestión de energía con flexibilidad centrada en el usuario) con el que poder controlar los dispositivos, basado en el sistema de prioridades comentado (AESP), utilizando para la transmisión tecnologías WiFi y ZigBee, permitiendo cambiar la prioridad a los electrodomésticos y minimizar el consumo de energía y la interacción humana.

La siguiente investigación a comentar, hace hincapié en la reducción del consumo en el modo standby que poseen los dispositivos electrónicos. Se ha demostrado que un 10% del total de la energía gastada en una casa es debido a este modo de inactividad [2].

Para ello, esta investigación sugiere el uso de un enchufe que vigila periódicamente el consumo de potencia a través de un "monitoring circuit" como se muestra en la siguiente figura:

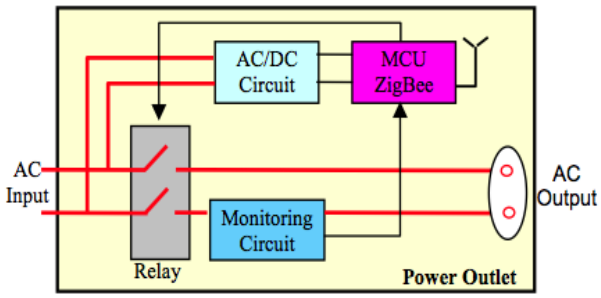

Figura 2: Monitoring circuit

El funcionamiento del enchufe es el siguiente: Inicialmente se fija un umbral en la memoria del microcontrolador. Cuando la potencia medida sea inferior al umbral (modo stand-by) durante un corto periodo de tiempo (por ejemplo 2 minutos), el relé de dentro del enchufe cortara el suministro y se ahorrara energía, pasando al estado de OFF, en donde no se vigila la potencia consumida, quedando a la espera de que un comando de despertar le llegue a través de una comunicación ZigBee, con la que activará el relé para suministrar de nuevo la energía en el enchufe.

Cuando el electrodoméstico que se enchufa cambia, el umbral es reconfigurado al medir los niveles de lo que está consumiendo y de lo que consume en modo stand-by.

El siguiente experimento es una mezcla entre las dos investigaciones anteriores. Esta investigación se centra en los dispositivos de iluminación de un casa, aunque podría ser extendido a cualquier otro aparato electrónico en un hogar.

El experimento consta de un "Smart Controller", instalado entre un enchufe y tres lámparas (para el ejemplo llevado a cabo en la investigación) de distintas potencias, lámpara incandescente (60W), halógeno (50W) y un tubo fluorescente (32W).

El controlador maneja el suministro de energía del enchufe en función del precio de esta, el cual esta introducido en el ordenador. Si el coste que está produciendo el dispositivo es alto, entonces se corta el suministro. La lámpara podrá volver a su funcionalidad cuando el precio se haya reducido después de un cierto tiempo [3].

Lo que primero se hace es averiguar una estimación del uso en el hogar para un determinado día. En la siguiente gráfica se muestra este patrón de uso:

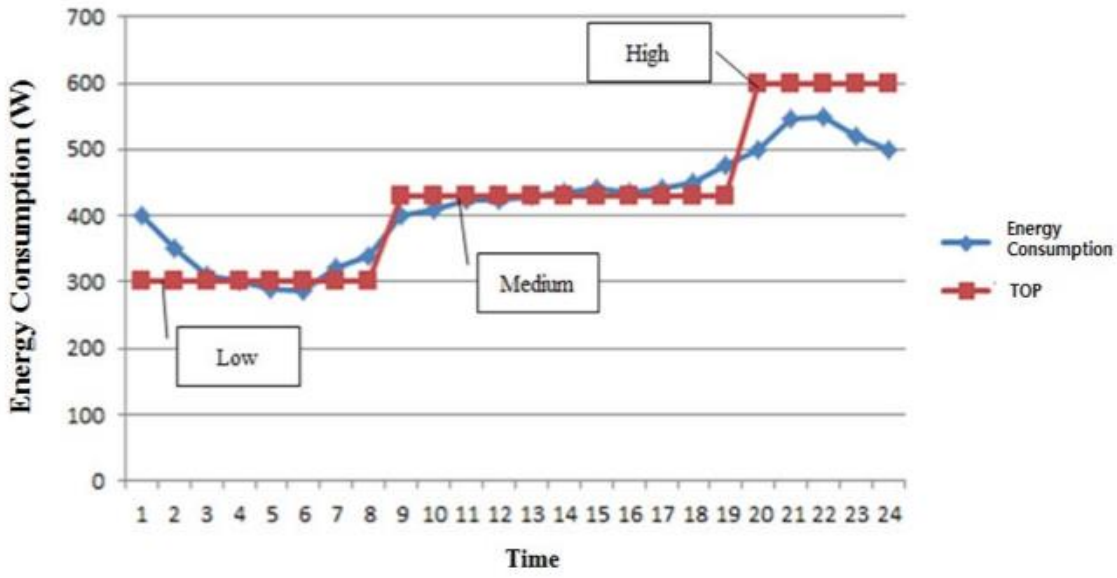

Figura 3: Ejemplo de la evolución del consumo durante un día completo

Como se puede observar se han definido tres zonas diferentes en función de la de energía consumida (Low, Medium, High), en donde es lógico aplicar, que cuanto mayor sea el consumo mayor sea el precio.

En el primer estado, en Low, cualquiera de las tres lámparas podría ser usada. En el estado intermedio la lámpara incandescente seria apagada debido a que es la que más gasta, pudiendo usarse las otras dos restantes. Y por último en High la única lámpara que se podría usar sería el tubo fluorescente ya que es la que menos consume.

Por último comentar en esta sección que algunos ayuntamientos de España se están planteando la instalación de este tipo de sistemas. Un ejemplo es el ayuntamiento de Marbella en el que se instalarán sensores de consumo para controlar el consumo eléctrico en redes públicas [5].

Se trata de un equipo electrónico, basado en la colocación de baterías de condensadores, que permite conocer cada quince minutos cual es el consumo real de la dependencia donde se instala el sensor.

El ayuntamiento de Marbella prevé ahorrar con este sistemas 100.000€/año. Además, como dato curioso, la empresa encargada del suministro eléctrico al ayuntamiento de Marbella, Endesa, es la responsable de la implantación de las baterías de condensadores.

2.2. Tecnologías

En este apartado se muestran las posibles tecnologías que se pueden usar para la transmisión inalámbrica de la información recogida por el sensor de corriente al servidor:

2.2.1. ZigBee

ZigBee es un protocolo que se utiliza en muchas redes domésticas. El protocolo es desarrollado y mantenido por un consorcio de más de 300 empresas llamadas ZigBee Alliance. ZigBee está basado en el estándar IEEE 802.15.4 de redes inalámbricas de área personal (Wireless Personal Area Network, WPAN). Su objetivo son las aplicaciones que requieren comunicaciones seguras con baja tasa de envío de datos y maximización de la vida útil de sus baterías.

ZigBee se ejecuta en la parte superior del 802.15.4 ocupando la capa de red, la capa de transporte, y un conjunto de interfaces en la capa de aplicación. En la figura 4 se muestra el protocolo ZigBee en la pila OSI [5].

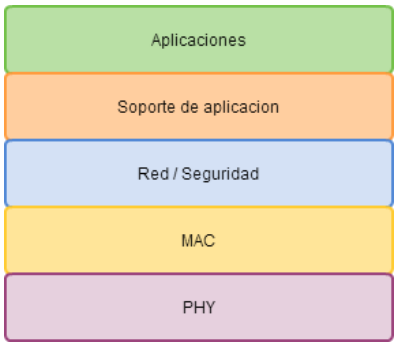

Figura 4: Pila OSI - ZigBee

ZigBee ha sido desarrollado específicamente para proporcionar consumos de energía bajos, fácil integración (se puede fabricar con poca electrónica), velocidades de transmisión de hasta 250kbps y con un rango de cobertura de 10 a 75 metros. La larga duración de la batería es debido a los largos ciclos de trabajo que usa y al multihop routing. Este tipo de enrutamiento permite que cada paquete

pueda viajar sobre distancias relativamente grandes, mientras que cada dispositivo por sí mismo sólo necesita transmitir en distancias cortas.

La capa de red ZigBee está basada en el estándar 802.15.4 anteriormente comentado que permite crear topologías en forma de estrella, árbol, o la más usada, la red en malla ya que permite que si en un momento dado, un nodo del camino falla y se cae, pueda seguir la comunicación entre todos los demás nodos debido a que se rehacen todos los caminos.

Cabe destacar que la ZigBee Alliance´s Core Stack Working Group está trabajando en la generación de un nuevo protocolo ZigBee enfocado para aplicaciones domésticas como medidores inteligentes (ya sean medidores eléctricos, de agua o gas). Este nuevo protocolo usará iPv6.

Posibles aplicaciones que pueden ser llevadas a cabo usado ZigBee quedan reflejadas en la siguiente imagen:

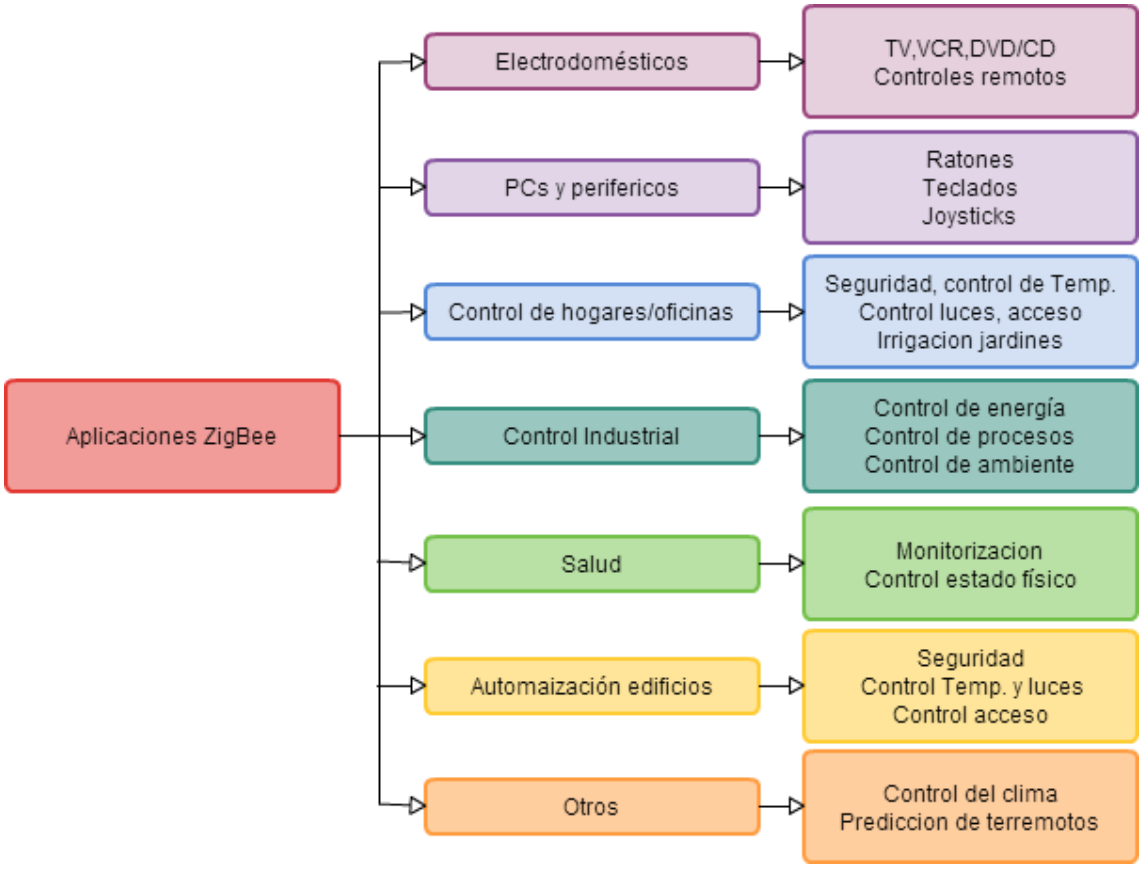

Figura 5: Ejemplos de aplicaciones con la tecnologia ZigBee

Y por último, en cuanto al futuro de ZigBee se espera que los módulos ZigBee se conviertan en los transmisores inalámbricos más baratos del mercado (en el entorno de 6 euros), y que además sean producidos de forma masiva.

2.2.2. Bluetooth

Es mucho más conocido que el protocolo anteriormente comentado.

Es utilizado para conexiones a corta distancia de voz, datos y audio, de forma inalámbrica en la banda de los 2.4 GHz.

En la figura 6 se puede observar las velocidades de transmisión y el rango de distancia que alcanzan las distintas tecnologías.

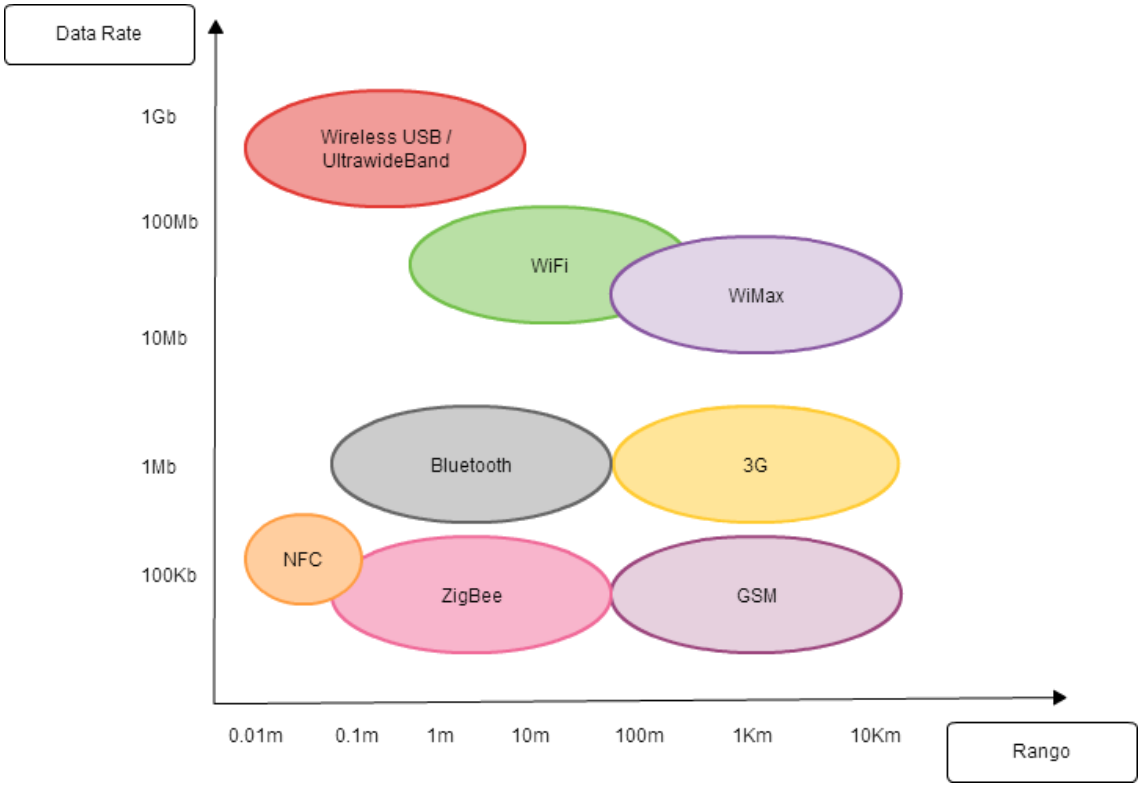

Figura 6: Data rate vs distancia

Bluetooth soporta el protocolo IP. Bluetooth es usado para comunicaciones entre pocos dispositivos, velocidades mayores que ZigBee y que no necesiten tantas restricciones de consumo. Además, ofrece la posibilidad de crear pequeñas redes inalámbricas y facilita la sincronización de datos entre equipos [4].

Los dispositivos que utilizan Bluetooth requieren periódicamente activarse y sincronizarse con un dispositivo maestro. Esto puede llevar alrededor de 3 segundos mientras que para ZigBee es del orden de milisegundos.

2.2.3. ZigBee vs Bluetooth

ZigBee es similar a Bluetooth pero con algunas diferencias notables resumidas en los siguientes cuatro puntos [6]:

- Una red ZigBee puede constar de un máximo de hasta 65.535 nodos distribuidos en subredes de 255 nodos, frente a los ocho máximos de una subred Bluetooth.

- ZigBee tiene un menor consumo eléctrico que Bluetooth. ZigBee tiene un consumo del orden de 30 mA transmitiendo y de 3 μA en reposo, frente a los 40 mA transmitiendo y 0,2 mA en reposo que tiene Bluetooth. Este menor consumo se debe a que el sistema ZigBee se queda la mayor parte del tiempo dormido, mientras que en una comunicación Bluetooth siempre se está transmitiendo y/o recibiendo.

- ZigBee tiene una velocidad de hasta 250 kbps, mientras que la de Bluetooth es de hasta 3.000 kbps.

- Debido a las velocidades, cada uno es apropiado para unos servicios determinados. Por ejemplo, mientras que Bluetooth se usa para aplicaciones como la telefonía móvil, la velocidad de ZigBee se hace insuficiente, siendo usado en aplicaciones como la domótica, sensores médicos, y en artículos de juguetería, en los cuales la velocidad de transferencia de datos es menor.

2.2.4. WiFi

WiFi o también conocido por IEEE 802.11 (WLAN, Wireless Local Area Network), se incluye en casi todas las redes domésticas. Por el momento, es el protocolo más aceptado para las comunicaciones inalámbricas, aunque por lo general WiFi requiere más energía que Bluetooth o cualquiera de los protocolos basados en la familia 802.15.4.

En la actualidad podemos encontrarnos con muchos tipos de comunicación WiFi: 802.11b, que emite a 11 Mb/seg u 802.11g mucho más rápida, a 54 Mb/seg.

La velocidad y alcance (~100-150 metros) que WiFi ofrece le convierten en una combinación perfecta para el acceso a internet inalámbricamente [5].

Al tratarse de conexiones inalámbricas, no es difícil que alguien interceptase la comunicación y que obtuviera acceso a nuestra información. Por eso, es recomendable la encriptación de la transmisión para emitir en un entorno seguro.

Con WiFi esto es posible gracias a WPA (WiFi protected access), mucho más seguro que su predecesor WEP. WPA añade nuevas características de seguridad, como la generación dinámica de la clave de acceso.

Cabe comentar que WiFi no es compatible con otros tipos de conexiones inalámbricas como Bluetooth, GPRS, etc…

En la tabla I se muestra un análisis comparativo entre las tres tecnologías inalámbricas anteriormente presentadas [7]:

	ZigBee (802.15.4)	Bluetooth Low Energy	WiFi (802.11)
Velocidad de datos	250Kb/s	1Mb/s	+1Mb/s
Rango	10-20m	5-15m	1-100m
Potencia	Baja	Baja	Media
Vida de la batería	Año	Semana	Día
Banda de frecuencias	2.4GHz, 869MHz, 915MHz	2.4GHz	2.4GHz
Tiempo de conexión	30ms – 1s	Hasta 10s	Hasta 3s
Aplicaciones	Electrodomésticos Control de la luces Seguridad del hogar	Salud/Monitorización Relojes Teclados/ratones	Audio digital Voz/Video Networking

Tabla 1: Comparativa entre distintas tecnologías inalámbricas

De la comparativa entre los diferentes estándares de comunicación más populares que comparten la banda de 2,4 GHz, se ha decidido usar WiFi.

Se ha elegido porque posee mayor velocidad de transmisión, mayor rango de cobertura y por su movilidad, permitiendo que un usuario pueda conectarse a Internet desde cualquier punto del área de cobertura de una red WiFi.

Además permite el acceso de múltiples dispositivos en una red, es decir que permite establecer una "red de dispositivos", la cual no es posible por ejemplo con Bluetooth, ya que solo permite unir dispositivos por pares

También cabe comentar desventajas que ofrece WiFi. La principal se encuentra en el consumo de energía, que es bastante más elevado que ZigBee y Bluetooth.

2.2. Medidores de consumo eléctrico en el mercado actual

¿Qué se puede hacer con estos aparatos?

Con estos medidores se puede hacer un seguimiento del consumo eléctrico a nivel individual de los dispositivos ya sean electrodomésticos, ordenadores,

periféricos,… o del seguimiento del consumo a nivel global de la electricidad usada en un hogar.

Las mediciones permiten saber el consumo en cada momento, consumo mínimo y/o máximo, la obtención de un histórico y además se puede incluir las tarifas del proveedor de electricidad y por lo tanto poder conocer en cada momento el impacto económico de lo que se consume o además poder conocer el impacto medioambiental.

Estos aparatos permitirán saber si se está malgastando energía, en que aparatos o en que momentos, y por tanto se podrá decidir con esa información donde se puede ahorrar y reducir la factura, hacer un seguimiento para comprobar el consumo y así poder llevar un control del gasto, del consumo y/o del impacto que tienen para el entorno.

Son extremadamente fáciles de instalar (son como un enchufe-temporizador), los cuales podemos cambiarlos de ubicación cuando queramos, y así poder comprobar el consumo de los principales electrodomésticos del hogar, aunque sea por tandas.

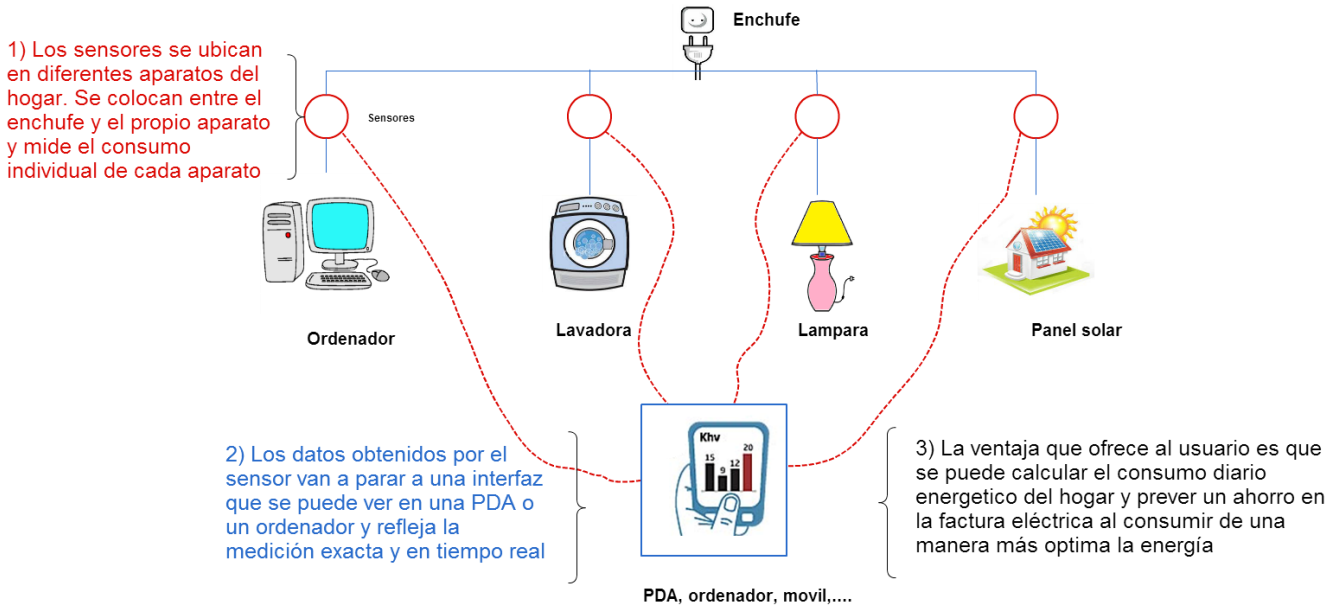

Figura 7: Ejemplo con sensores de corriente

Existen 2 tipos de medidores: los medidores de consumo individual y los de consumo total.

2.2.1. Medidores de consumo individual

Los monitores individuales se conectan directamente a un enchufe y permiten conectar en ese enchufe los aparatos a los que se quiere hacer un seguimiento. De esta manera se puede ver cuánto consume un aparato individualmente o un grupo de aparatos o electrodomésticos.

Es muy útil si se quiere conocer el consumo en particular de un aparato y una forma de identificar cuáles son los electrodomésticos que más consumen según los hábitos de uso en una casa. Ésta es la mejor forma de identificar los consumos de los electrodomésticos en espera (stand-by) y que pueden llegar a ser muy altos.

En cambio, no te permiten saber el consumo de todo lo que está conectado completamente a la red eléctrica de la casa y lo que no está conectado a enchufes (como por ejemplo la iluminación de la casa) por lo que habrá un consumo que no se podrá medir con estos medidores.
Ejemplos de uso:

- Medición del consumo del ordenador y periféricos y su consumo en stand-by.

- Medición del consumo individual de cada electrodoméstico: lavadora, frigorífico, lavavajillas, etc…

- Seguimiento del consumo de los aparatos de audio y video en el salón.

Algunos modelos de medidores de consumo individual son:

2.2.1.1. Energy monitoring socket

Con tan sólo con conectar el dispositivo al enchufe en la pared y luego el aparato que se quiera medir, se tiene instalado un medidor de electricidad. Se requiere también que se introduzca el precio por unidad de energía que se paga a la compañía eléctrica para así poder mostrarlo gráficamente en la pantalla que incorpora [8].

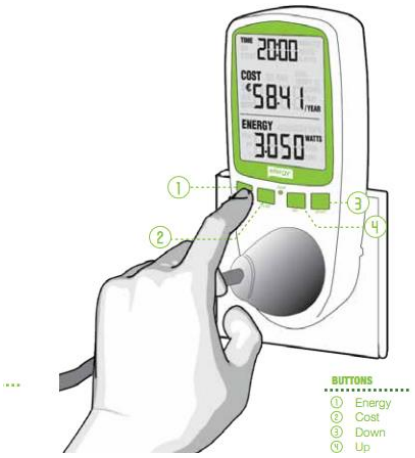

Figura 8: Sensor Efergy

http://efergy.com/es/

El precio aproximado de este dispositivo de medida es de 25€.

2.2.1.2. Belkin

2.2.1.2.1. Conserve insight

Permite controlar el impacto de la energía que consume en la factura eléctrica y en el medio ambiente. En un solo display se pueden ver los vatios, el coste de funcionamiento y las emisiones de CO2 derivadas del consumo eléctrico.

Sería un equivalente al Energy monitoring socket 2.0 de Efergy, mostrado anteriormente con la nueva incorporación de las emisiones de CO_2 a un precio aproximado de 27€ [9].

Figura 9: Sensor Conserve insight

https://www.belkin.com/conserve/insight/

2.2.1.3. Cliensol energy ESI (enchufe sensor inalámbrico).

Figura 10: Sensor ESI

http://efergy.com/es/products/energy-saving-products/remotecontrolledsocket-twosocket

Otro modelo, con las mismas características que los dos productos anteriormente citados, con la salvedad de la pantalla. La información es transmitida inalámbricamente y el consumo es mostrado en un display. En dicha pantalla se podrá observar tanto la variación del consumo, como por ejemplo, encender o apagar un electrodoméstico o una lámpara [10].

También se podrá observar mediante un gráfico el consumo de la mañana, tarde y noche, la hora y la temperatura ambiente.

Los datos registrados se recogen y pueden ser descargados mediante diferentes programas al PC para obtener un registro del consumo de energía.

El precio aproximado de este dispositivo de medida es de 26€.

2.2.2. Medidores inalámbricos del consumo total en el hogar

La función de estos dispositivos es medir el consumo eléctrico total en la vivienda, para que se pueda conocer no sólo la cantidad de energía consumida, sino también las horas y las actividades que más gasto suponen en el hogar.

Normalmente se componen de un pequeño sensor ubicado en el cuadro eléctrico. En la mayoría de ellos se trata de un clip (pinza) que se engancha al cable de fase. Éste recoge la información sobre la cantidad de electricidad consumida y la envía a una pequeña consola que muestra los datos recopilados, convirtiéndolos por ejemplo en medidas útiles como los kilovatios o la cantidad en euros gastada.

Dependiendo del modelo, se puede consultar además del consumo instantáneo, la energía gastada durante el último día, la de la última semana, la del último mes, la energía media por día, etc… Además, la mayoría incluyen la función de descargar los datos al ordenador y consultarlos más en detalle, con gráficas que ilustran el consumo y mucha otra información de utilidad.

Los modelos más recientes y avanzados se conectan a internet y envían la información a una plataforma Web que interpreta los datos recopilados, pudiendo consultarlos desde cualquier ordenador o incluso desde las aplicaciones que los fabricantes ofrecen a sus clientes. Con esto, los usuarios podrán informarse en tiempo real del consumo energético de los electrodomésticos de la casa y por tanto detectar fallos de funcionamiento. Por ejemplo, se tendría la posibilidad de rebajar la potencia de un microondas si el aparato está consumiendo más de lo necesario.

Estos dispositivos, cuentan entre sus opciones con la posibilidad de poder sugerir al usuario el modo de optimizar el uso de los aparatos y electrodomésticos en el hogar. Por ejemplo el sistema podría recomendar, sobre la base de los datos registrados, la posibilidad de apagar la lavadora en el caso de que esté consumiendo más energía de la necesaria.

Estos dispositivos de control energético permitirán a los hogares evitar un derroche innecesario en el uso de los aparatos eléctricos, al mismo tiempo que reducirían la factura de la luz.

2.2.2.1. Sensor de consumo total Efergy:

2.2.2.1.1. Elite classic

En la figura 11, se pueden observar tres dispositivos con los que cuenta este kit para poder obtener el consumo eléctrico.

El primero a comentar, dispositivo en la esquina inferior izquierda, corresponde con el sensor (gancho) que deberá conectarse al cable en el cual se quiere hacer la medición de corriente.

Una vez hecha la medición (cada 6 segundos), ésta será enviada al display mediante un transmisor (dispositivo del medio) de forma inalámbrica.

Y por último en el display se podrá observar las mediciones realizadas [8].

Figura 11: Sensor Elite classic

http://efergy.com/es/elitev1-monitor

Funciones y características del Efergy elite classic:

- Selecciona hasta 2 tarifas diferentes.
- Alerta de audio cuando se excede un límite preestablecido.
- Fácil de instalar, configurar y usar.
- Consulta de hasta tres modos de información: coste, CO2 y kWh.
- Comprueba el consumo en tiempo real.
- Descubre el consumo histórico diario, semanal o mensual.
- Averigua el consumo promedio por hora, semana o mes.
- Hasta 70 metros de alcance.
- Posibilidad de alimentación por vía eléctrica a través de un adaptador.

- Vida útil de hasta 12 meses (con pilas alcalinas de 2400 mAh).
- Averigua y reduce la huella de carbono.
- Rango de transmisión: 40-70m
- Frecuencia:433MHz
- Tiempo de transmisión: 6 seg. 12 seg. ó 18 seg.

El precio aproximado de este dispositivo es de 55€.

A continuación se muestran un par de imágenes, figura 12, que recogen dos ejemplos de lo que se podría consultarse en el display con la información recogida por los sensores desde un ordenador.

En la primera imagen de la izquierda se muestra el histórico de un determinado día. Se puede observar como a las 15:00 se produjo un pico de consumo y como en las horas de descanso no se consumía nada. También se puede observar en la parte inferior el consumo total de energía, el coste que supuso y las emisiones de CO_2.

Figura 12: Ejemplos de históricos con la información recogida por el sensor de corriente

http://efergy.com/es/

En la imagen de la derecha se puede observar una comparativa de consumo entre dos dispositivos en el mes de julio.

2.2.2.1.2. Elite optical

El contador instantáneo de electricidad Elite Optical funciona con el contador con puerto óptico LED (en vez de un sensor con forma de gancho, ahora se tiene un sensor IR (infrared)) que lee el consumo energético cada 30 segundos para poder ver en el display por ejemplo el impacto de encender o apagar la luz, poner en marcha el aire acondicionado o usar cualquier otro electrodoméstico.

El precio aproximado de este dispositivo es de 70€ [8].

Figura 13: Sensor Elite optical

http://efergy.com/es/elitev1-monitor

2.3.2.2. EcoTouch

Figura 14: Sensor ecotouch

http://efergy.com/uk/products/electricity-monitors

Este kit está formado por los mismos componentes que el Elite classic más un conector llamado ecoTouch que se conectará en el enchufe en el cual vaya conectado el electrodoméstico a medir [8].

El funcionamiento del sensor es el mismo que el anteriormente comentado. Se trata de un sensor que se acopla al cable de fase. La información de consumo medida es enviada en tiempo real al monitor ecoTouch, teniendo las mismas funcionalidades que los dispositivos anteriores.

La novedad consiste en que conectando los electrodomésticos que se quiera controlar a los ecoSockets, se podrá conseguir encenderlos y/o apagarlos a distancia, a través del monitor (incluso se puede agrupar varios ecoSockets en habitaciones, haciendo más fácil el control energético).

El monitor se actualiza instantáneamente a medida que se consume la energía y además sirve para revisar el consumo histórico.

El precio aproximado de este dispositivo es de 60€.

Capítulo 3

Diseño de la solución propuesta

La idea de este proyecto reside en la toma de muestras de corrientes con sensores para poder llevar un control del consumo en los diferentes dispositivos eléctricos de un hogar.

A continuación se exponen la elección de los diferentes elementos que configuran la solución propuesta.

Elección del sensor

El sensor que se ha decidido usar en este proyecto ha sido el Waspmote de la marca Libelium. Se trata de un dispositivo sensorial de bajo consumo que permite la creación de redes sensoriales inalámbricas [11].

Es un dispositivo open source, que permite la creación de infinidad de aplicaciones y con un precio económico.

En el mercado se disponen de múltiples opciones, pero a un coste reducido, éstas se reducen a dos: ArduinoUno y Waspmote.

Estos dos dispositivos se basan en una interfaz de programación, creada inicialmente por Arduino, que da lugar a plataformas de hardware libre, basadas en una placa con un microcontrolador y un entorno de desarrollo diseñado para facilitar el uso de la electrónica en proyectos de diversos ámbitos.

ArduinoUno cuenta con un microprocesador ATmega328P en lugar del ATmega1281 que posee Waspmote.

Además la Waspmote incorpora la posibilidad de incluir una tarjeta microSD en el propio dispositivo, mientas que si se quiere realizar esta ampliación en ArduinoUno habría que comprar un módulo especial.

Tanto Waspmote como Arduino soportan diferentes protocolos de transmisión de datos. ArduinoUno necesita un módulo especial de interconexión donde insertar las tarjetas de comunicación lo cual no ocurre en la Waspmote, donde se pueden conectar directamente los módulos en la propia tarjeta.

Se ha decidido usar la Waspmote de Libelium ya que estaba disponible en la escuela. Además se trata de una versión más actual que la de Arduino para la creación de software/hardware de libre distribución.

A continuación se puede observar una fotografía de una Waspmote en la que se puede destacar su reducido tamaño.

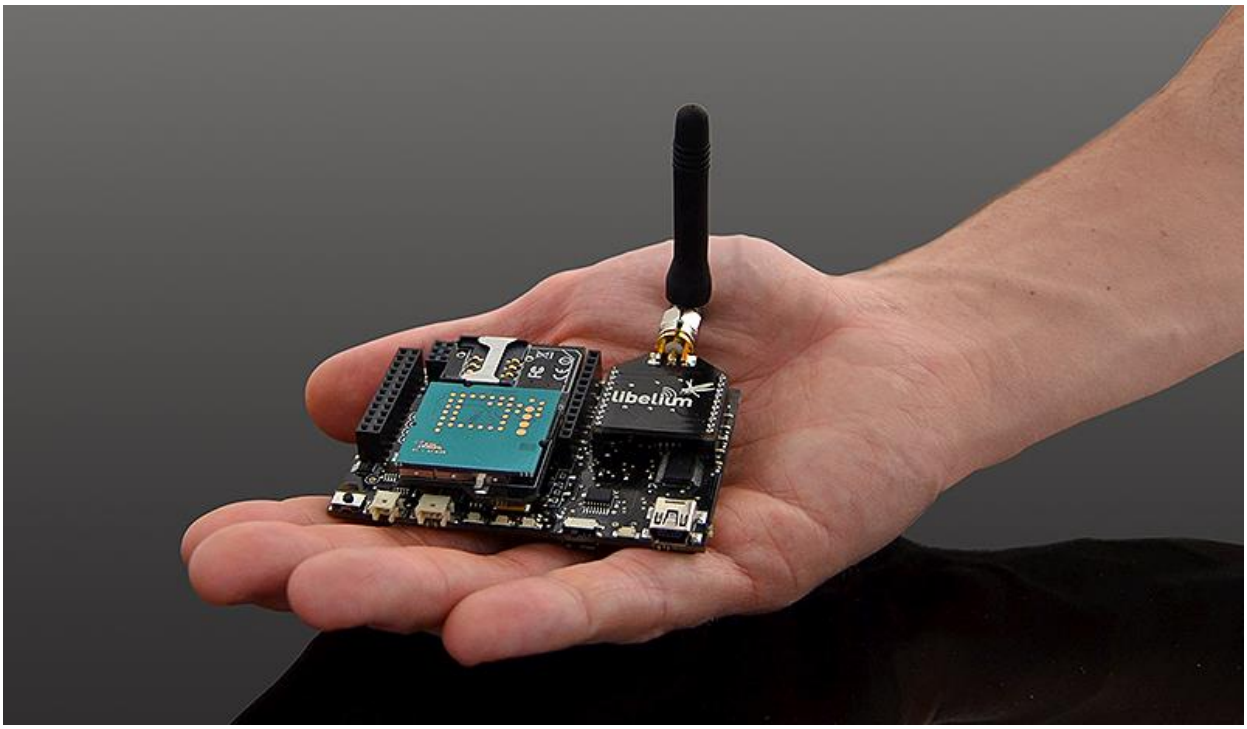

Figura 15: Waspmote

Por último se van a citar las ventajas más reseñables con las que cuenta la Waspmote:

- Una arquitectura modular la cual permite añadir funcionalidades como:
    - Uso de una tarjeta microSD.
    - Comunicación Wi-Fi.
    - Comunicación Bluetooth y ZigBee.
    - Comunicación GPRS.

  - GPS.
  - …

- Programación en lenguaje derivado del lenguaje C, mediante una interfaz Java.
- Gran cantidad de recursos online.
- Código de libre distribución.
- Dispositivo móvil.
- Bajo consumo y alta duración de la batería.

Elección de la base de datos

Una vez se han tomado las muestras, éstas serán enviadas al servidor vía WiFi para guardarlas en una BBDD.

De entre las bases de datos disponibles se ha elegido MySQL ya que cuentas con las siguientes características [12]:

- Software open source
- Alto rendimiento en la gestión, dando una alta velocidad de procesado
- Posibilidad de ser ejecutada en sistemas sin grandes requerimientos tanto de velocidad como de memoria
- Facilidad de configuración e instalación
- Soportada en una gran variedad de Sistemas Operativos
- Baja probabilidad de corromper datos, incluso si los errores no se producen en el propio gestor, sino en el sistema en el que está
- Su conectividad, velocidad, y seguridad hacen de MySQL Server altamente apropiado para acceder a bases de datos en Internet

Elección de la aplicación móvil

Se hará uso de una aplicación móvil para poder consultar el consumo actual de un determinado electrodoméstico de forma remota.

Actualmente existen diversos sistemas operativos para dispositivos móviles. En la tabla siguiente puede observarse una comparativa entre ellos [13] [14] [15].

Características / Sistemas	IOS	Android 4.0	Windows
Kernel	OS X	Linux	Windows CE 7
Estándares soportados	GSM y CDMA	GSM y CDMA	GSM y CDMA
Multitarea	Si	Si	Si
Cortar copiar pegar	Si	Si	Si
Hardware soportado	iPhone, iPad y iPod	Amplia variedad	Variedad limitada
Compatibilidad en modelos	Compatibilidad total	No hay compatibilidad	Compatibilidad total
Seguridad	Muy seguro	Susceptible de malware	Muy seguro
Tienda de películas	iTunes	Si (desde el market)	Zune
Integración social media	Facebook y Twitter	Facebook y twitter	Facebook, twitter, Windows Live
Music store	iTunes	Google Play Music	Zune
Juegos	Amplia variedad	Poca variedad	XBOX Live
Social gaming	Game center	No	Bing
Navegador	Safari	Chrome	Internet Explorer
Motor de búsqueda por	Google	Si	Si
Productividad	iWork	Google Docs	Office Mobile
Reconocimiento de voz	Si	Iconos de Widgets	Si
Sincronización WiFi	Si	Solo mediante terceras aplicaciones	Si
Apps	+500.000	+250.000	+30.000
Soporte Cloud	iCloud	Google sync	skyDrive
Personalización	Limitada	Si	No
Soporte flash	No	Si	No

Tabla 2: Comparativa entre sistemas móviles

Aunque básicamente se ha elegido Android por tratarse de un sistema operativo de código abierto y mayor disponibilidad de encontrar dispositivos Android para implementar una aplicación.

3.1. Descripción funcional del sistema

El objetivo de este proyecto fin de carrera es la toma y almacenamiento, tanto local como remoto, de muestras del consumo de corriente de los dispositivos eléctricos de un hogar con la ayuda de sensores Waspmote.

En periodos de tiempo configurables, el sensor tomará muestras de la corriente consumida por el dispositivo a analizar, procediendo a almacenarlas en la memoria

SD de la Waspmote. Al alcanzarse un número predeterminado de muestras, se trasmitirán mediante paquetes de datos a un servidor.

Si el envío de información resulta satisfactorio, se borrarán todas las muestras almacenadas en la tarjeta SD y volvería a repetir el proceso.

Si por el contrario no se recibiese contestación por parte del servidor de haber recibido correctamente los paquetes enviados, se procedería a realizar una retransmisión de los mismos durante un número programado de intentos. Si tras dichos reintentos no se consiguiese establecer la conexión, las muestras obtenidas permanecerán almacenadas en la tarjeta SD y la Waspmote seguiría tomando muestras y guardándolas a continuación de las anteriores en modo FIFO. Cuando volviese a llegar al umbral, enviaría todo lo que tuviera almacenado y volvería a repetirse el proceso de forma repetitiva.

En la parte del servidor, todas las muestras que le llegan serán almacenadas en un base de datos, liberando por tanto la memoria SD del sensor y estando disponibles para poder dar un amplio uso a posteriori de las mismas por medio de los informes emitidos por solicitud del usuario.

La interfaz de usuario de la aplicación del servidor dispondrá de distintas opciones entre las que cabe destacar:

- Poder conocer el consumo instantáneo del dispositivo.
- Poder cambiar el tiempo de adquisición de las muestras, periodo de muestreo.
- Conocer consumos medios en un determinado periodo temporal.
- Obtener un gráfico histórico con todas las muestras almacenadas hasta el momento por un determinado sensor.
- Almacenar dicho histórico en un fichero dentro del ordenador.

Además el proyecto también incluye el desarrollo de una aplicación para dispositivos móviles, realizada en Android, que permitirá consultar el consumo actual de un dispositivo de forma remota mediante una conexión de datos con el servidor, el cual, a modo de dispositivo puente, solicitará a la Waspmote conocer dicho valor, quedando éste registrado en la base de datos.

En la siguiente figura se puede observar las diferentes partes que intervienen en la realización de este proyecto:

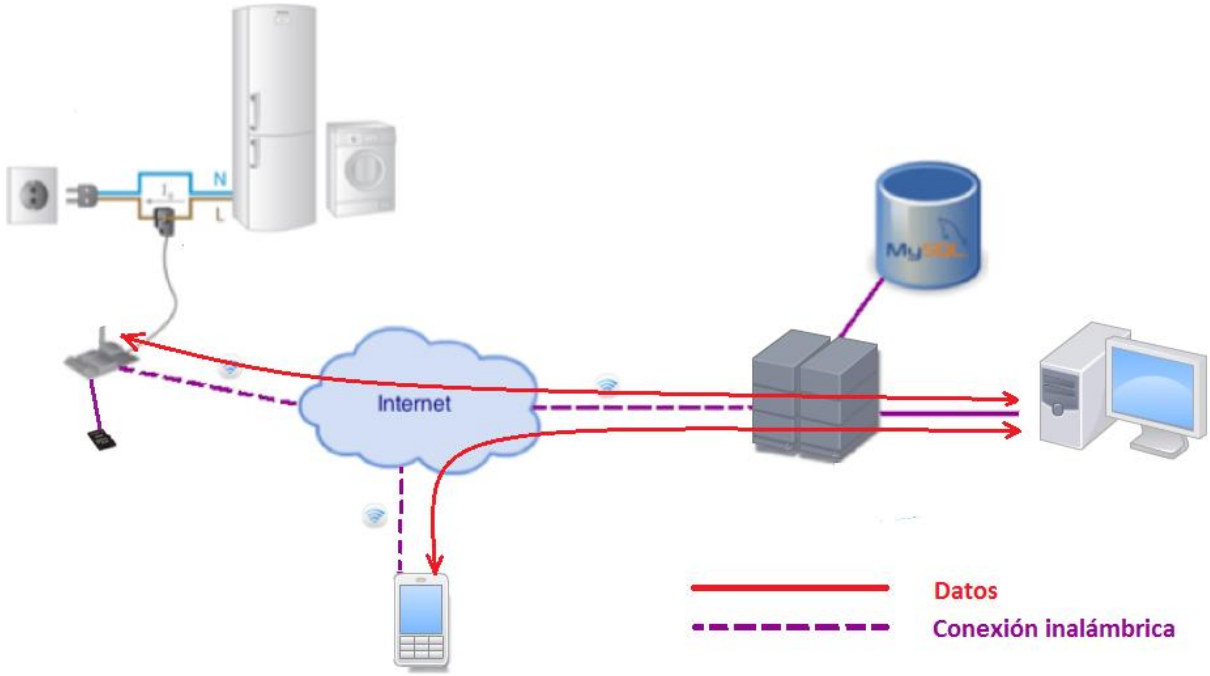

Figura 16: Escenario final del proyecto

3.2. Entorno de trabajo

A continuación se van a enunciar los distintos programas que son necesarios para poder llevar a cabo este proyecto.

Waspmote IDE

Es el software necesario para poder programar la Waspmote junto con todos los módulos asociados, para permitir la adquisición de las muestras de corriente, su almacenamiento en una memoria SD y el posterior envío y recepción de tramas al servidor [16].

Se puede descargar gratuitamente en la página web de Libelium. Permite su instalación en sistemas operativos Linux, Windows y Mac.

Se debe de programar en C y cuenta con una interfaz Java para visualizar determinados contenidos de la Waspmote.

MySQL

MySQL es un sistema de gestión de bases de datos relacional, multihilo y multiusuario [12].

MySQL está escrito en C y C++ y destaca por su gran adaptación a diferentes entornos de desarrollo, permitiendo así su interacción con otros lenguajes de

programación como Java, Perl, PHP,… y su integración en distintos sistemas operativos (Windows, Mac,…).

Cabe también señalar la característica open source de MySQL que hace que su utilización sea gratuita e incluso que se pueda modificar con total libertad, pudiendo descargar hasta su código fuente. Esto ha favorecido muy positivamente en su desarrollo y frecuentes actualizaciones, para hacer de MySQL una de las herramientas más utilizadas en la programación orientada a Internet.

Android Studio

Es el software que se usará para la creación de la aplicación con la que se podrá obtener en un dispositivo móvil de forma remota el consumo instantáneo de un determinado dispositivo eléctrico.

Android Studio es un entorno de desarrollo integrado para la plataforma Android. Está disponible para desarrolladores en sistemas Windows, Mac OS X y Linux.

Como lenguaje de desarrollo se usa Java para construir las aplicaciones. Además se usa el "metalenguaje" XML para el diseño de interfaces a través de Parsing. También se necesitaran las herramientas y librerías, es decir el SDK de Android [17].

3.3. Módulos

A continuación se va a proceder a comentar en detalle los módulos anteriormente citados para la realización del proyecto: Waspmote, servidor y aplicación en Android.

3.3.1. Waspmote

Básicamente el software implementado en la Waspmote consta de dos partes, una función de inicialización o **setup()** y el método **loop**().

La primera se trata de la función **setup**(). Se aplica en la primera parte del código que se va a ejecutar. En esta parte se incluye la inicialización de los módulos que se van a usar, como también aquellas partes del código importantes cuando la Waspmote empiece a funcionar.

Y por otro lado, está la función **loop**(), cuyo objetivo en este proyecto es la de medir el consumo de corriente cada cierto tiempo, guardar dichas muestras en la memoria SD y la de enviar y recibir tramas de información al servidor.

A continuación se puede observar un diagrama de bloques que muestras las acciones que serán llevadas a cabo en cada uno de los dos métodos anteriormente comentados.

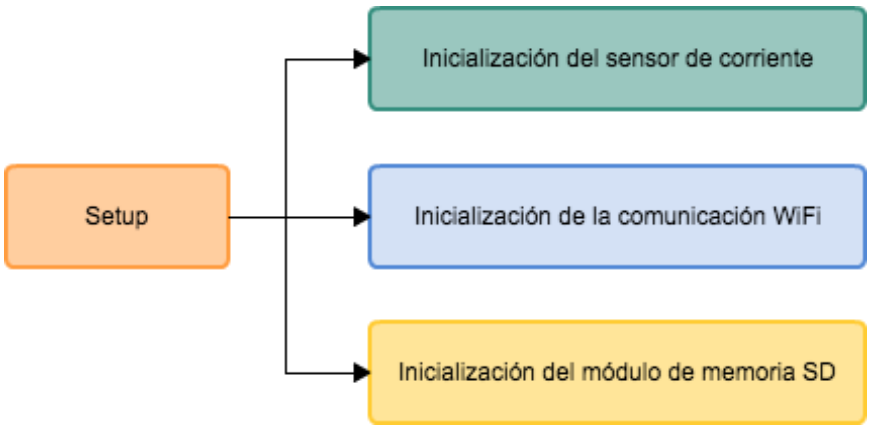

Figura 17: Diagrama de bloques del método setup

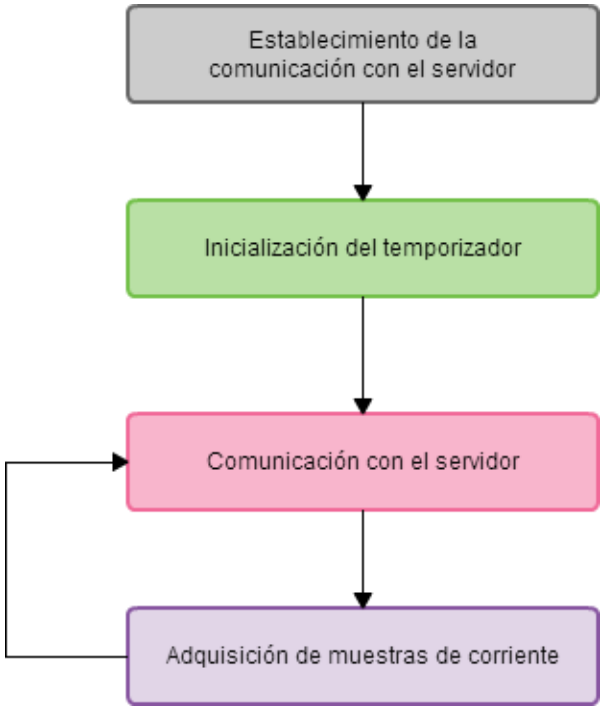

Figura 18: Diagrama de bloques del método loop()

Inicialización del sensor de corriente

En este apartado se inicializa la Waspmote, indicándola que tiene asociado un sensor de corriente para realizar la toma de muestras de corriente.

```
SensorSmartv20.ON();
```

Inicialización de la comunicación WiFi

En esta sección se realiza la inicialización del proceso de comunicaciones WiFi entre el dispositivo y el servidor.

Este proceso conlleva las siguientes etapas:

1. Reconocimiento y activación de la antena WiFi.

   Se define en cuál de los dos sockets que posee la Waspmote se conectará la antena WiFi, bien el socket0 o el socket1. Para la realización de este proyecto se ha elegido el socket0. Nótese que se podría haber usado el socket1.

2. Elección de la red WiFi y contraseña de acceso.

3. Configuración del protocolo de transporte.

   En principio la Waspmote permite la utilización de diversos protocolos (UDP, TCP, FTP, HTTP, etc…).

   Para la comunicación entre la Waspmote y servidor se ha decidido utilizar sockets, pudiéndose hacer que dos programas, situados en diferente máquinas, pueden intercambiarse cualquier flujo de datos [18].

   Para la realización de esta parte se la decidido realizarlo mediante sockets UDP (User Datagram Protocol). Esto es debido a que UDP es un protocolo sencillo y sin necesidad de conexión. Dicha capacidad de transferencia sin conexiones de UDP le hace útil para servidores que reciben una gran cantidad de peticiones pequeñas de un alto número de clientes.

   Además es útil para aplicaciones que necesitan una transmisión rápida y efectiva, ya que a pesar tener comprobación de errores no tiene opción para recuperar o corregir los mismos.

   Por el contrario, no garantiza la fiabilidad de entrega de los paquetes, por lo que en el proyecto se ha optado por comprobar que cada vez que se enviaba un mensaje, le tenía que llegar una confirmación. Si ésta no llegaba, se volvía a intentar hasta un máximo de 3 veces.

   Otra ventaja de UDP respecto a TCP, es que con UDP se puede enviar un mensaje a varios receptores a la vez, mientras que en TCP, al haber una conexión previa establecida, sólo se puede enviar el mensaje al receptor que está conectado al otro lado. Esta ventaja puede resultar muy provechosa para futuras mejoras como el poder solicitar el consumo eléctrico de todos los dispositivos eléctricos que se hayan conectado al servidor.

Para la creación del socket se ha necesitado tener en cuenta los dos requisitos siguientes:

- Ambos programas (Waspmote y servidor) deben de localizarse mutuamente, ya sea dentro de una red local o de Internet.

- Ambos programas deben de poder intercambiarse información.

Por tanto para poder realizar la comunicación con el servidor es necesario que se conozca la dirección de red IP. Además se debe saber el número del puerto, que se asigna a un programa dentro de un ordenador

Cabe destacar que al no contar con una dirección IP pública, con la que poder acceder al servidor desde cualquier lugar en cualquier momento, en la realización de este proyecto se ha accedido desde dentro de la red WiFi local, es decir, con direcciones IP privadas del tipo 192.168.1.XXX.

Y por último, en cuanto a la elección del puerto se ha optado por elegir el 8440 al no estar ocupado previamente por ningún programa ni estar reservado para otros protocolos.

4. Configuración de cómo resolver las direcciones IP.

   Para la realización de este proyecto se ha decidido que la Waspmote use el protocolo DHCP (Dynamic Host Configuration Protocol). Se trata de un protocolo de red que permite a los clientes obtener sus parámetros de configuración automáticamente. Se trata de un protocolo de tipo cliente–servidor en el que generalmente un servidor posee una lista de direcciones IP dinámicas y las va asignando a los clientes conforme éstas van quedando libres, sabiendo en todo momento quién ha estado en posesión de esa dirección IP, cuánto tiempo la ha tenido y a quién se la ha asignado después.

Inicialización del módulo de memoria SD

El primer paso en esta sección es activar el módulo de memoria con ayuda del comando SD.ON(). Dicha sentencia es necesaria para indicarle a la Waspmote que este módulo va a ser utilizado a lo largo del proyecto.

Por otro lado, en esta misma sección se borrará todo el contenido que tuviera almacenado la tarjeta SD, para así asegurarse de que está totalmente vacía.

Además, se creará el archivo de trabajo en el que se almacenarán las muestras de corriente justo después de ser tomadas por el sensor con ayuda de la sentencia SD.**create**(“pfc.txt”).

También es en esta sección donde se define el límite de muestras que serán almacenadas en la memoria para que sean enviadas en un trama. Para la realización de este proyecto se ha elegido escoger un valor de 25 muestras a enviar por mensaje.

Una vez se han realizado todas las inicializaciones se va a proceder a comentar el resto del programa.

Establecimiento de la comunicación

Lo primero que se hace es activar el módulo WiFi y conectarse a la red WiFi seleccionada previamente en la inicialización. Para ello se usa la función WIFI.**join**(RedWiFi) que recibe como parámetro el nombre de la red WiFi a la que la Waspmote se tiene que conectar como una cadena de caracteres y devuelve un boolean en función de si la conexión se ha realizado satisfactoriamente o no.

Si la conexión no se ha podido llevar a cabo, debido a que el router este caído u otros fallos, se volverá a intentar todas las veces necesarias hasta que se consiga.

En el caso que se haya podido conectar a la red, el siguiente paso obligatorio es la creación de la conexión UDP, mediante la siguiente función:

```
WIFI.setUDPclient(DIRECCION_IP, PUERTO_REMOTO, PUERTO_LOCAL)
```

En la que se indican la dirección IP del servidor, con el que se intercambiará mensajes y los puertos remoto y local de la conexión a usar.

Al igual que ocurría con la función de conexión a una determinada red inalámbrica, esta función devolverá un boolean en función de si la creación de la conexión UDP se ha realizado con éxito o no. En caso de que no se consiga establecerlo se volverá a repetir el proceso hasta que finalmente se consiga, incluido el método anteriormente comentado, WIFI.**join**(RedWiFi).

Por el contrario, si se consigue crear la conexión con servidor, el siguiente paso es mandar un mensaje al servidor para establecer la comunicación. Para ello se envía un mensaje al servidor con ayuda de la siguiente función:

```
WIFI.send("conecta");
```

La función send() es la responsable de enviar cualquier mensaje al servidor.

Al tratarse del establecimiento de la comunicación, se ha decido enviar un primer mensaje con la palabra conecta, para indicarle al servidor de que se trata del primer mensaje.

Una vez el servidor reciba el mensaje, lo procesará y enviará de vuelta a la Waspmote un mensaje conteniendo la fecha y hora actuales del sistema. Dicho mensaje supondrá la confirmación por parte del servidor al mensaje enviado por la Waspmote y a su vez permitirá a la Waspmote conocer la hora y fecha actuales

para poder saber el instante concreto en que las muestras de corriente sean tomadas.

Para realizar la sincronización de la fecha y hora en la Waspmote se ha creado el método **actualizarFecha_Hora**(), que recibe por parámetro la fecha y la hora enviadas por el servidor.

A continuación se procesa y se crea un string con dicha información para poder pasárselo por parámetro a la función **setTime**(), responsable de establecer la fecha y hora en la Waspmote. A continuación se muestra un ejemplo para poder fijar la fecha y la hora en la Waspmote:

```
tiempo=”15:10:20:03:17:35:00”;
RTC.setTime(tiempo);

15: año.
10: mes.
20: día.
03: día de la semana.
17: hora.
35: minuto.
00: segundo.
```

Una vez actualizado el sistema, el siguiente paso es proceder a poner un temporizador para que cada cierto tiempo la Waspmote tome una muestra de corriente y así pueda llevar a cabo una correcta monitorización del consumo eléctrico, con ayuda del método **ponerAlarmaEnSegundos**().

En cuanto al valor de dicho temporizador, se ha decidido inicializarlo a 10 segundos, pudiendo ser modificado por el usuario en cualquier momento.

Para poder explicar dicho método, es importante conocer la siguiente información de antemano. La Waspmote cuenta con dos alarmas, alarma1 y alarma2. Alarma 1 es usado cuando se requiere una precisión de fecha/horas/minutos/segundos mientras que alarma2 se usa para una precisión de fecha/horas/minutos. Para la realización de este proyecto se ha optado por el uso de la alarma 1 para que así se puedan fijar temporizadores que vayan desde horas hasta alarmas programadas para pocos segundos.

La función encargada de poner el temporizador para la alarma1 es: **RTC.setAlarm1**(**tiempo**, **offset**, **modo**). Esta función cuenta con 3 parámetros de entrada.

- **tiempo**: representa el nuevo periodo de muestreo a cambiar como una cadena de texto.

- **offset**: representa uno de los dos modos que se pueden seleccionar, offset o absolute. Si se selecciona offset, el parámetro tiempo es añadido al tiempo actual en el RTC como alarma. Por el contrario, si se elige el modo absolute, es el parámetro tiempo el que actúa como tiempo para la alarma.

- **modo**: representa los diferentes modos con los que cuenta la función en cuanto a la precisión para que la alarma se active. Existen seis modos en función de la exactitud requerida:

    - RTC_ALM1_MODE1: Coincidencia por día, horas, minutos y segundos

    - RTC_ALM1_MODE2: Coincidencia por fecha, hora, minutos y segundos

    - RTC_ALM1_MODE3: Coincidencia por horas, minutos y segundos

    - RTC_ALM1_MODE4: Coincidencia por minutos y segundos

    - RTC_ALM1_MODE5: Coincidencia solo por segundos

    - RTC_ALM1_MODE6: Coincidencia cada segundo

Una vez comentado el significado de los distintos posibles valores que se le pueden pasar a esta función, se va a proceder a comentar los valores elegidos.

Con respecto al segundo parámetro, se podría usar cualquiera de los dos, pero por simplicidad, se ha optado por el de offset, ya que simplemente es indicarle el tiempo restante para que salte la alarma.

Y por último referente al modo, se ha decidido optar por la modalidad **RTC_ALM1_MODE3** que cuenta con una precisión de horas/minutos/segundos, ya que para la realización de este proyecto no ha surgido la necesidad de tener una fecha concreta.

Por otro lado comentar que cada vez que se genera una alarma, se activará el flag **intFlag**, indicando que el suceso ha ocurrido. Por tanto si se quiere realizar una acción cuando la alarma haya ocurrido, es necesario comprobar dicho flag.

Una vez ambos dispositivos han establecido la conexión, se entrará en un bucle en el cual se estará comprobando si ha llegado un mensaje procedente del servidor o si por el contrario le toca adquirir una muestra si ha pasado el tiempo necesario.

En la figura 19 se puede observar el diagrama de flujo que ilustra este proyecto.

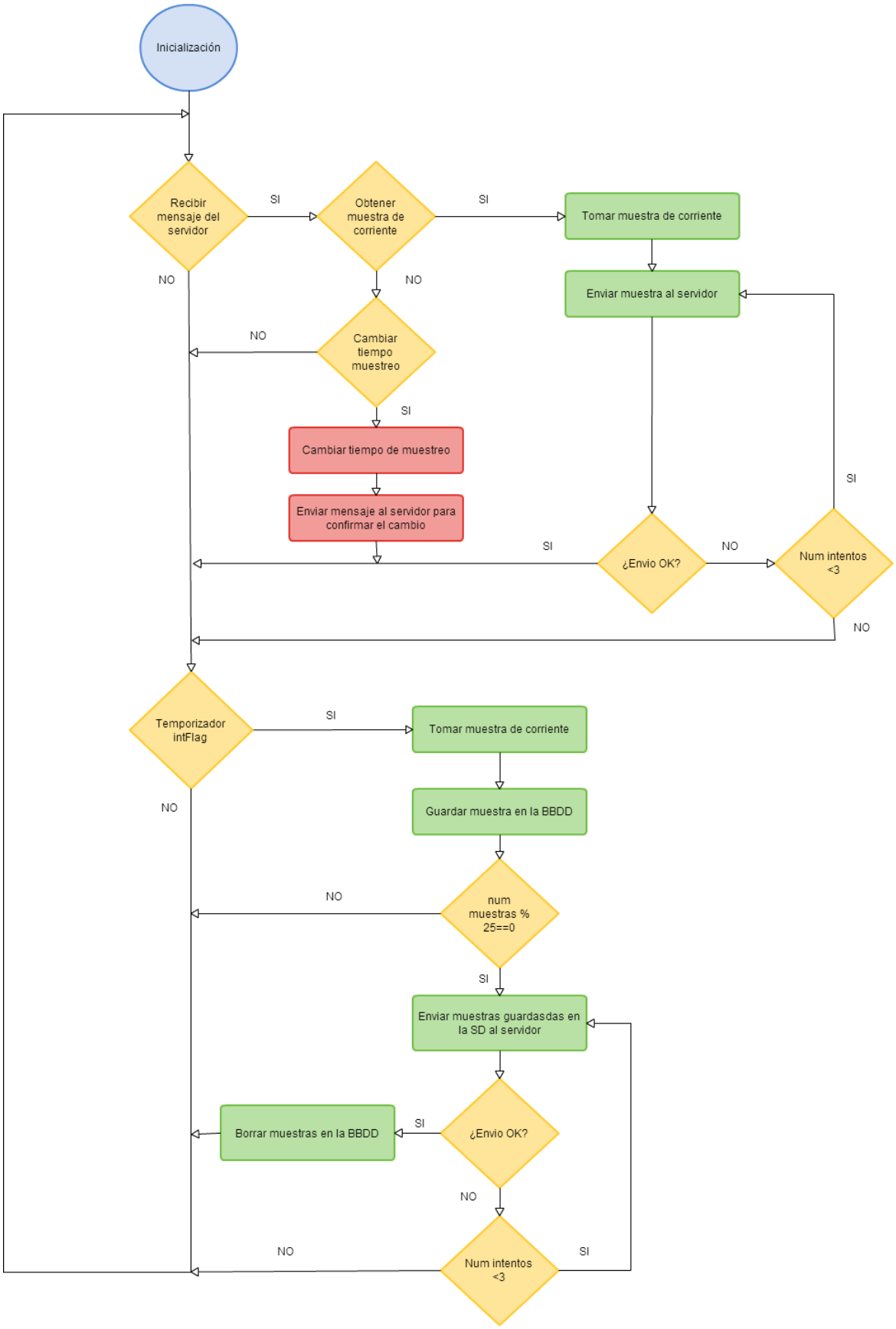

Figura 19: Diagrama de flujo de la Waspmote

A continuación se procederá a explicar el diagrama anterior.

Comunicación con el servidor

Dicha sección se centra en la recepción de tramas por parte del servidor. Para ello se ha creado un método llamado **recibirRespuesta**() en el que se comprueba si le ha llegado un mensaje por parte del servidor.

Para ello se utiliza la siguiente sentencia

```
WIFI.read()
```

Esta función se encarga de la recepción de tramas. Dicha función necesita recibir por parámetro el tipo de espera que necesita de realizar la función, NOBLO o BLO, (no bloqueante y bloqueante).

La opción BLO se trata de una espera bloqueante, esto es que la función **read**(), se quedará esperando, hasta que llegue una trama. Mientras que el modo NOBLO, se quedará también esperando pero hasta un máximo de dos segundos.

Para la elección de este proyecto se ha elegido la opción NOBLO, ya que la otra opción tiene el grave problema de que si no recibe nada, se quedaría bloqueado y no se podrían realizar más acciones.

Una vez se recibe un mensaje procedente del servidor, el siguiente paso es procesar dicha información para saber qué tipo de información está solicitando el usuario o bien si se trata de algún tipo mensaje el cual confirma alguna trama.

Para ello se guarda en una variable el mensaje que se recibe del servidor y se compara con una de las siguientes posibilidades.

- Si el contenido enviado por el servidor empieza por '1', se procede a llamar a la función **menu1**(). Usar esta función significa que el usuario ha solicitado conocer el consumo actual del dispositivo eléctrico.

  Para ello se procede a llamar a la función **medirSensor()** que devolverá el valor solicitado. Dicha función será llamada cada vez que se quiera conocer lo que está consumiendo el dispositivo eléctrico.

  Para ello es imprescindible que cada vez que se quiera tomar una muestra se active el sensor de corriente y se espere un pequeño tiempo para que éste se estabilice y que así se pueda calibrar adecuadamente para su correcta utilización.

  A continuación se procede a usar la función que permite leer la corriente que circula por el sensor:

SensorSmartv20.readValue(**SENS_SMART_CURRENT**),

A esta función hay que pasarle como parámetro único el tipo de sensor que va a utilizarse (corriente, humedad, temperatura, audio, etc….), en nuestro caso **SENS_SMART_CURRENT**, que hace referencia a la corriente. Y devolverá el valor de la corriente como un float de 4 decimales.

Una vez obtenido el valor de la muestra de corriente se necesitará rellenar una cadena de texto con el valor pedido en amperios para enviárselo al servidor.

El modo de construcción de dicha cadena es el siguiente:

- Se empieza rellenándolo con los caracteres "$1", para indicar que el mensaje que se va a enviar, contiene una sola muestra. Esta información le resultará muy útil al servidor para procesar dicho mensaje

- A continuación se añade el símbolo "&" que indica que a continuación viene el valor de la muestra

- Después se adjunta el valor obtenido por el sensor de corriente seguido de sus unidades en amperios, A

- Y por último la fecha y hora de la obtención de la muestra

Una vez creado el string, éste se le pasa como parámetro único a la función **send**(), para que se lo envíe al servidor.

Después se debe de esperar un breve tiempo para que llegue una confirmación por parte del servidor que indique que el envío de información se ha realizado correctamente. Si dicha confirmación llega, se dará como satisfactoria la conexión, sin embargo si no llega durante un pequeño periodo de tiempo se procederá a volver a enviarlo hasta un máximo de 3 repeticiones, dejando un breve espacio temporal entre estos envíos por si llegase la confirmación.

- Si el contenido recibido en la Waspmote empieza por '2', se llamará a la función **menu2**().

  Esta función se encarga de modificar el tiempo con el que se tomarán las muestras. Para ello filtra el contenido del mensaje recibido para obtener el tiempo solicitado (dicho tiempo puede ir desde unos pocos segundos hasta días).

- Si el contenido recibido es igual a "ACK", significa que la conexión se ha realizado con éxito. Esto es muy útil para saber si los mensajes enviados por la Waspmote al servidor se ha realizado satisfactoriamente.

Si por el contrario no llega ningún mensaje o no contiene ninguno de los mensajes anteriormente citados, se devolverá -1 para indicarlo.

El siguiente paso que tendrá que ser llevado a cabo en el código, será comprobar si la alarma ha saltado, verificando si se ha activado el flag, **intFlag**, y si es así se procederá a la toma de una muestra de corriente. Si por el contrario no se ha activado se volverá a repetir otra vez los pasos anteriormente comentados.

Es importante comentar porque se ha decidido realizarlo de esta manera.

La solución idónea hubiera sido que la Waspmote estuviera en modo de hibernación (ahorro de energía) para reducir el consumo y sólo se despertase cada vez que tuviera que tomar una muestra de corriente, y además pudiera ser factible el recibir mensajes del servidor mientras la Waspmote se encuentra en el este estado.

Esto no es posible ya que mientras la Waspmote se encuentra en este estado no puede recibir vía WiFi, no siendo posible el realizar peticiones desde el servidor, dado que la Waspmote no las detectaría. La Waspmote no identifica los mensajes recibidos vía WiFi como interrupciones.

Adquisición de muestras de corriente

Si efectivamente se ha producido la activación del flag, **intFlag**, se procederá a la obtención de una muestra de corriente mediante la función **medirSensor()**, anteriormente comentada.

La idea consiste en que una vez se ha tomado la muestra de corriente, si el número de muestras guardadas en la memoria es inferior a 25, límite fijado para la realización de este proyecto, se procede a guardar la muestra en la memoria SD justo a continuación de las anteriores. Si por el contrario se llega al umbral de 25, se procederá a enviar todas las muestras almacenadas hasta el momento en una sola trama.

Para ello se ha decido realizar el almacenamiento de las muestras en la memoria SD por líneas de tal manera que por cada línea en la memoria se almacenen 5 muestras junto con sus fechas y horas de obtención. Se ha realizado de esta forma ya que se trata de la longitud máxima de caracteres que se puede devolver usando las funciones que proporciona la Waspmote.

A continuación se muestra el formato (cadena de caracteres) con el que se ha decidido almacenar una muestra de corriente en la memoria SD.

```
strcpy(aux2,"&");
strcat(aux2,valor_muestra);
strcat(aux2,"A ");
strcat(aux2,RTC.getTime());
strcat(aux2," ");
```

Primeramente se crea un string con el valor "&" que indica el principio de la cadena en la que se guardará la muestra.

A continuación se le añade el valor de la muestra de corriente, valor_muestra. Para ello es necesario utilizar la función **float2String**() que permite realizar un casting de float a string para poder guardarlo correctamente como una cadena de texto. Posteriormente se añade el carácter "A" que indica las unidades de la muestra en amperios seguido de la fecha y hora de obtención de dicha muestra.

Tras guardar dicha muestra en la tarjeta de memoria, el siguiente paso es comentar como se guardan dichas muestras en la memoria SD.

Para guardar una muestra de corriente hay que comprobar primeramente el número de muestras que hay almacenadas en la memoria SD. Si dicho número es cero, es decir que todavía no hay ninguna muestra almacenada, la primera muestra se escribiría al comienzo de la primera línea con ayuda de la función SD.**append**().

Si por el contrario hubiera más de una muestra, el siguiente paso es comprobar el número de muestras que hay en la fila. Si este número es inferior a cinco, se procedería a añadir la nueva muestra a continuación de la anterior con ayuda del comando.

```
SD.writeSD(nombreFichero, muestra, posicion);
```

A dicha función hay que pasarle como primer parámetro el nombre del fichero en donde se quiere guardar la muestra. Como segundo término el valor de la muestra a almacenar y por último la posición en la que la queremos guardarla.

Si por el contrario el número de muestras almacenadas por fila ha alcanzado el límite de 5 muestras, se procederá a repetir el proceso anteriormente comentado pero en una nueva fila.

Una vez que se llegue al límite de muestras comentando con anterioridad, 25, se procedería a mandar las muestras almacenadas hasta el momento en una única trama junto con la fecha de adquisición de cada una de ellas. Para ello se necesita realizar los siguientes pasos:

Como primer paso se extrae la información de las 25 muestras guardadas en la memoria una tras otra. Para ello se crea una cadena de caracteres cuyo primeros

caracteres contienen el número de muestras que se enviarán en la trama, precedido del carácter '$'. Para una trama con 25 muestras quedaría "$25". De esta forma se le indica al servidor el número de muestras que se están enviando y así éste pueda interpretarlas correctamente cuando las reciba.

Para poder ir extrayendo las muestras de la memoria se ha creado un pequeño bucle el cual irá recorriendo todas las filas de la memoria y guardando dichas muestras en la variable a enviar al servidor

Una vez creado la trama a enviar, se procede a llamar a la función WIFI.**send**(), para enviársela al servidor.

Al tratarse de sockets UDP, hay que garantizar el envío de información. Para ello se ha optado por enviar el mensaje y esperar un máximo de dos segundos para que le llegue una confirmación del envío con la opción NOBLO, espera no bloqueante.
Si llega un mensaje de confirmación, significa que el servidor ha recibido las muestras correctamente y a continuación se procede a reiniciar las variables involucradas. Además se borrarán todos los archivos que hubiera hasta el momento, para asegurarse de que la memoria está totalmente vacía y es seguro el poder ir completando la memoria con los próximos nuevos valores tomados por la Waspmote.

Si por el contrario no llega la confirmación pasados los dos segundos, se volverá a reintentar el envío dos veces más por si se tratase de algún fallo técnico como: caída del router, de la línea, etc…. Si dichos dos intentos no son satisfactorios, es decir que no se ha conseguido establecer conexión con el servidor, las muestras se mantendrían guardadas en la memoria y se volverá a repetir el código explicado en esta sección y la próxima vez que se tengan que enviar al servidor, se transmitirán todas aquellas que no se hubieran podido entregar más las nuevas 25 muestras, enviándose ordenadamente en el tiempo.

3.3.2. Servidor

El servidor nos permitirá por medio del menú del interfaz de usuario la configuración de diferentes opciones de funcionamiento así como poder manejar las comunicaciones inalámbricas con el dispositivo portátil Waspmote.

Para la creación del servidor se ha creado una clase inicial llamada PFC_main.java con un sólo método **main**(), en el que se crearán instancias a dos clases.

Las dos instancias a comentar en esta sección son **InterfazMenu** y **ComunicacionWaspmote**.

La primera de ella, **InterfazMenu**, se centra en mostrar un menú por pantalla con distintas opciones. Entre ellas destacan el conocer el consumo actual de un determinado dispositivo eléctrico o la de cambiar el tiempo de adquisición de las muestras.

Por otro lado, **ComunicacionWaspmote** se basa en la recepción de tramas con información del consumo eléctrico o bien el envío de tramas a la Waspmote con alguna de la opciones elegidas por el usuario a través del menú.

Cabe destacar que todas las muestras recibidas en el servidor serán almacenadas en una base de datos para poder, a partir del procesamiento de las mismas dar una amplia información del consumo del dispositivo bajo estudio.

A continuación se puede observar un diagrama de bloques con las acciones más relevantes llevadas a cabo por el servidor:

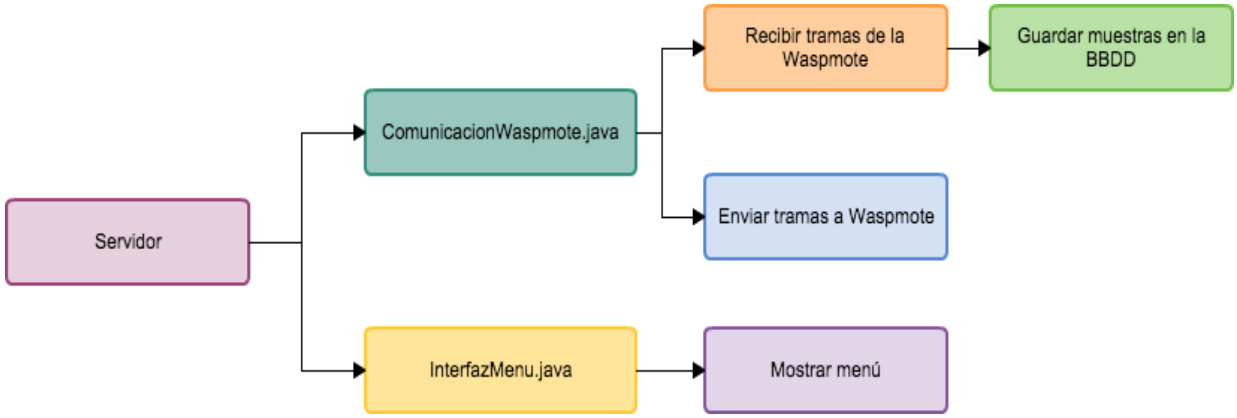

Figura 20: Diagrama de bloques del servidor

Ambas clases extienden de la clase Thread para que así se puedan ejecutar varias acciones de forma simultánea en el servidor, como por ejemplo que por un lado se esté mostrando un menú con diversas opciones y mientras en un segundo plano se puedan recibir tramas con muestras de corriente, analizarlas y guardarlas en la BBDD.

Antes de comentar dichas clases, es preciso comentar la creación de la base de datos, la cual será mencionada posteriormente.

Dicha construcción se centra básicamente en crear una tabla en MySQL, llamada valoresCorrientes, que almacene los siguientes tres valores.

```
mysql> describe valoresCorrientes;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| id    | varchar(20) | YES  |     | NULL    |       |
| valor | varchar(10) | YES  |     | NULL    |       |
| fecha | varchar(20) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
```

Figura 21: Configuración de la BBDD

Dónde:

- Id hace referencia a la dirección IP del dispositivo que mando la información
- Valor, es el consumo eléctrico en amperios de un determinado dispositivo
- Fecha, indica cuando fue tomada la muestra de corriente por la Waspmote

Una vez puestos en antecedentes, se procederá a explicar la realización del servidor.

ComunicacionWaspmote

Esta clase se centra en enviar y recibir datos de la Waspmote. Para ello cuenta con un método **run**(), el cual se repite indefinidamente.

En dicho bucle se llevarán a cabo las siguientes tareas:

- Reconocimiento de la Waspmote

  Se realiza cada vez que una nueva Waspmote quiera establecer conexión con el servidor. Para ello se llama a la función **recibir**(), la cual devolverá un string con el mensaje enviado por la Waspmote. Acto seguido se necesita hacer una mera comprobación booleana de si el mensaje que llega contiene la palabra "CONECTA" o no.

  Si resulta que no se trata del primer mensaje por parte de la Waspmote, se ignoraría y se continuaría con el resto del código. Si por el contrario resulta que sí se trata del primer mensaje, se procederá a enviar al dispositivo que solicita conexión, la fecha y hora actuales del sistema para que se sincronice, con ayuda de las siguientes sentencias:

```
Date fecha = new Date();
String fecha_string = "**" + fecha.toString();
```

  Nótese que a la hora de crear la cadena de texto se ha añadido un doble asterisco para que así la Waspmote pueda reconocer el tipo de mensaje cuando lo vaya a procesar.

  A continuación se muestra dicho intercambio de tramas entre ellos:

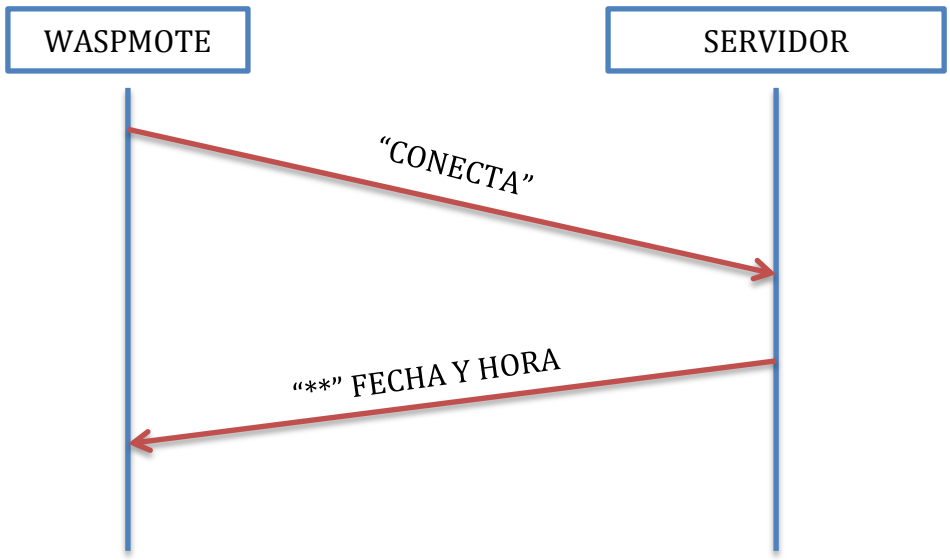

Figura 22: Intercambio de tramas incial

Una vez se ha establecido la conexión con la Waspmote, se mostrará un menú en la pantalla en el cual se podrá interactuar con la Waspmote (explicado con más detalle en la siguiente sección InterfazMenu).

- Recibir trama con muestras

  Cada vez que se recibe un mensaje que empiece por el carácter '$' significará que dicho mensaje contiene información con una o varias muestras de corriente.

  Para saber el número de muestras que contiene dicho mensaje habrá que procesar dicho mensaje. Dicha información vendrá justo a continuación del carácter dólar, $.

  Ante la correcta recepción de un mensaje, se enviará un mensaje de vuelta con la cadena de texto "ACK", la cual indica que el mensaje ha llegado correctamente.

  Una vez se ha enviado el mensaje, se procede a la extracción de las muestras de una en una. Es importante resaltar que las muestras vendrán acompañadas de su fecha de obtención y todas tendrán la misma longitud por lo que se facilita su extracción.

  A continuación se muestra el ejemplo de una trama con 2 muestras de corriente:

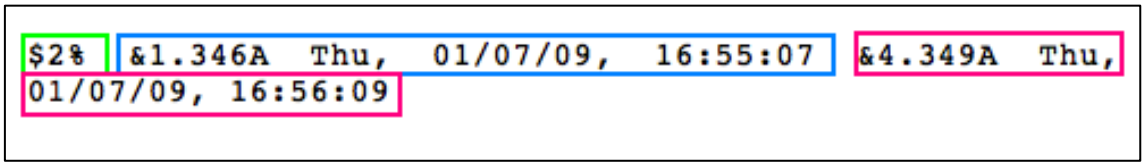

Figura 23: Ejemplos de una trama con dos muestras de corriente

Cada vez que se obtiene una muestra de corriente junto con su fecha y hora de obtención, se procede a almacenar en la base de datos junto con la dirección IP del dispositivo que las envió.

Para poder introducir dicha información en la BBDD se ha creado el siguiente método:

escribirDatosMySQL(String direccionIP, String mensaje_recibido, String fecha)

Cada vez se quiera usar la BBDD se necesita establecer una conexión con ésta. Para ello se debe de usar un driver (o conector). Este driver es una clase de Java que sabe cómo "dialogar" con la base de datos.
A continuación se muestran las líneas de código necesarias para realizar dicha unión con la base de datos:

```
import java.sql.Connection;
import java.sql.DriverManager;

...

try
{

  Class.forName ( "com.mysql.jdbc.Driver");
  DriverManager.getConnection (
"jdbc:mysql://localhost/proyectoFinCarrera",  "root",  "password ");

  ...
```

Nótese que dichas sentencias están dentro de un bloque try-catch debido a que pueden devolver un error si no se puede acceder a la BBDD.

El siguiente paso en este método es comentar como introducir los datos en la BBDD. Para ello la forma más directa para insertar una nueva fila en una tabla es mediante una sentencia INSERT. Se debe indicar la tabla a la que se quieren añadir las muestras, y los valores de cada una de éstas.

Los parámetros que necesita dicha función han de ser pasados entre comillas sencillas o dobles. (Para los datos numéricos no es imprescindible, aunque también pueden estar entrecomilladas).

```
executeUpdate("INSERT INTO valoresCorrientes (id,
valor, fecha) VALUES
('"+dirección_IP+"','"+valor_muestra
+"','"+fecha+"')");
```

Dicha función vuelve a tener que estar contenida en un bloque try-catch ya que puede generar error, como consecuencia por ejemplo de que el servidor MYSQL puede estar caído o si se introducen datos que no existe.

Y por último al tratarse de un método con una conexión abierta con la base de datos, es necesario cerrarla una vez se ha realizado la inserción de datos con ayuda del comando **close**().

A continuación se muestra una captura de pantalla en la que se puede observar cómo queda guardada la información en la BBDD de una muestra en las tres siguientes columnas.

```
+--------------+--------+--------------------+
| id           | valor  | fecha              |
+--------------+--------+--------------------+
| 192.168.1.33 | 3.762  | 15/02/22, 23:14:58 |
```

Figura 24: Ejemplo muestra guardado en la BBDD

- Otra de la acciones que se debe llevar a cabo es la de comprobar si tiene que enviar a la Waspmote una opción del menú elegida por el usuario

  Para ello, en función de la opción elegida por el usuario en la clase Waspmote (explicado en la próxima sección), se procederá a enviar un mensaje u otro.

  Si la opción elegida es la primera, significa que el usuario quiere conocer el consumo actual del dispositivo a medir y para ello se enviará el siguiente mensaje a la Waspmote:

```
enviar(s, "1");
```

  Una vez la Waspmote tome la medida, se le enviará al servidor. Acto seguido el servidor la procesará, la mostrará por pantalla y la guardará en la BBDD para que dicho valor quede registrado.

  Si por el contrario se ha seleccionado la opción 2 significa que el usuario quiere cambiar el tiempo de muestreo en el dispositivo. Para ello el mensaje a enviar a la Waspmote será:

```
enviar(s, "2 " + tiempoMuestreo);
```

En ambas sentencias lo primero que se indica es el socket UDP, s, para poder realizar la conexión. A continuación se indica el número, 1 ó 2, para indicar la opción elegida por el usuario. En el segundo caso se añade un espacio en blanco más el tiempo de muestreo elegido por el usuario para cambiar, con el siguiente formato HH:MM SS.

A continuación se comentará la realización de los métodos **enviar**() y **recibir**() para poder conseguir realizar el intercambio de información entre el servidor y la Waspmote, correctamente.

La creación del método **enviar**(DatagramSocket s, String mensaje) es la siguiente:

Esta función recibe como primer parámetro un DatagramSocket s, que da soporte al socket para el envío y recepción de datagramas UDP.

DatagramSocket proporciona un constructor que toma un puerto como argumento, para que sea usado por los procesos que necesitan usar un puerto en particular (con DatagramSocket existe la posibilidad de escoger un constructor sin argumentos que permite al sistema escoger un puerto local libre). Estos constructores pueden lanzar una excepción del tipo SocketException si el puerto ya está en uso o si está reservado, por tanto se precisan sentencias try-catch.

Y como segundo parámetro, recibe el mensaje con el contenido que será enviado hacia la Waspmote.

Para poder llamar a dicho método, se ha necesitado usar la clase DatagramPacket. Los DatagramPackets son usados para implementar el servicio de entrega en la comunicación inalámbrica. Para ello cada mensaje es enrutado desde una máquina a otra en base a la información contenida en el paquete.

Los objetos del tipo DatagramPacket se pueden transmitir entre procesos cuando un proceso los envía y otro los recibe.

```
DatagramPacket paquete = new DatagramPacket(msg, msg.length,
recibido.getAddress(), recibido.getPort());
```

A dicho constructor hay que pasarle 4 parámetros. Los primeros dos parámetros son el mensaje a enviar y la longitud de éste. Para poder utilizar correctamente esta función, se le debe de hacer una casting al mensaje de string a bytes, con ayuda de la función **getBytes**().

Seguidamente, hay que pasarle como parámetros la dirección IP y un puerto destino para poder realizar la conexión con la Waspmote. Para ello se utiliza una

variable DatagramPacket llamada recibido, en el que se guarda la información del dispositivo Waspmote al que se quiere enviar el mensaje. Esto es así, ya que pueden existir diversos dispositivos Waspmote conectados al mismo servidor y se debe especificar a quién va dirigido. La elección de la Waspmote se realiza por medio del menú de usuario.

Por último hay que llamar a la función s.**send**(paquete) en la que se envía el objeto DatagramPacket con la información anteriormente comentada.

Para llevar a cabo la recepción de tramas se ha creado el método **recibir**() cuyas características más relevantes son las siguiente:

Para poder recibir un mensaje se hace uso del método:

```
s.receive(recibido);
```

Dicho método necesita ser usado con ayuda de un DatagramSocket y exige recibir un DatagramPacket, recibido, como parámetro en el que se almacenará el contenido recibido.

El siguiente paso es guardar dicho datagrama. Se ha creado la función **guardarDatagramPacket**(recibido) para poder guardar las distintas conexiones con los diferentes Waspmotes que pudiesen existir y poder distinguir entre todas ellas.

A continuación se muestra la manera de extraer la información en el string str. Para ello se pasan como parámetros el DatagramPacket que se usó en la función **receive**() junto con sus límites de inicio y fin:

```
String str = new String(recibido.getData(), 0, recibido.getLength());
```

Y por último, se devolverá dicho String.

InterfazMenu.java

La creación de un menú permite mostrar una lista con diferentes opciones que el usuario podrá seleccionar.

En la figura 26 se muestra una captura de pantalla con las distintas opciones que cuenta el menú, para una Waspmote identificada por su dirección IP.

Para su realización se ha creado el método **mostrarMenu**(), el cual mostrará las distintas opciones a elegir y devolverá un string indicando la opción elegida por teclado.

```
-----------------------------------------
|Elegir una opción para 192.168.1.45:   |
|                                       |
| 1) Mostrar consumo instantaneo        |
| 2) Cambiar tiempo de muestreo         |
| 3) Mostrar histórico                  |
| 4) Guardar histórico                  |
| 5) Consumo medio en un mes            |
| 6) Seleccionar waspmote               |
-----------------------------------------
```

Figura 25: Captura de pantalla del menú para una determinada dirección IP

En función de la opción elegida por el usuario se procederá a realizar una de las siguientes funciones:

- Si la opción elegida es "1) Mostrar consumo instantáneo", se procederá a notificar a la clase ComunicacionWaspmote dicha elección, y ésta se encargará de enviar un mensaje a la Waspmote solicitando dicha información, **enviar**(s,"1"), como se ha explicado en la sección.

  Una vez le llegue el mensaje de vuelta con la información solicitada, la almacenará en la BBDD y la mostrará por pantalla.

  A continuación se muestra una captura de pantalla en la que se ha solicitado conocer el consumo actual para una determinada Waspmote identificada por su dirección IP.

```
Ha introducido: 1
Obteniendo consumo instantaneo...
**
/192.168.1.33 ———— Dirección IP de la Waspmote
**
$1% &4.190A Tue, 01/07/14, 17:55:15
Valor actual: 4.190A
01/07/14, 17:55:15
```

Figura 26: Opción #1 del menú

  Además se puede observar la trama enviada por la Waspmote y a continuación dicha información filtrada.

- Si la opción tomada es "2) Cambiar tiempo de muestreo", se procederá a modificar la frecuencia con la que muestras de corriente deben de ser tomadas con el siguiente formato HH:MM:SS. Comentar también que se comprobará que se introduce un tiempo correcto. En caso de no serlo, se volvería a solicitar dicho tiempo.

A continuación se le pasarán ambas informaciones, opción y tiempo de muestreo, a la clase ComunicacionWaspmote.java para que envíe ambos datos a la Waspmote en un mensaje, **enviar**(s, "2 " + tiempoMuestreo);

Y por último se esperará a que le llegue un mensaje de vuelta, confirmando que el cambio de la frecuencia de muestreo ha sido realizado con éxito.

A continuación se muestra una captura de pantalla solicitando el cambio del tiempo de muestreo a dos minutos y medio para una Waspmote identificada por su dirección IP.

Además se muestra como una vez llega la trama de confirmación "ACK_Tmuestreo", se muestra un mensaje indicando que se ha conseguido realizar el cambio satisfactoriamente.

```
Ha introducido: 2
Introduzca un numero en con el siguiente formato HH:MM:SS
00:02:30
Ha introducido: 00:02:30

Cambiando periodo de muestreo...
**
/192.168.1.33
**
ACK_Tmuestreo
...cambio realizado correctamente
```

Figura 27: Opción #2 del menú

- Si se selecciona "3) Mostrar histórico", se presentará en pantalla un gráfico con todas las muestras almacenadas por orden cronológico.

  Para ello se recurre al método **leerDatosMySQL()** de la clase MySQL.java**,** que se encargará de extraer en un vector todas las muestras guardadas en la BBDD.

  El principio de creación de este método es igual que el ya comentado para introducir datos. Se necesita crear una unión con la BBDD mediante un driver y luego se le debe de indicar en que tabla buscar los datos.

  A continuación hay que indicar la query o consulta que se quiere hacer sobre la tabla de valores de corriente de la base de datos. Para este método en concreto se quiere devolver todos los valores, por tanto la consulta será:

```
executeQuery("select * from valoresCorrientes")
```

Una vez establecido el criterio de la consulta, el siguiente paso es recorrer toda la base de datos de uno en uno e insertar todas las muestras y fechas que va encontrando en un vector.

A continuación se muestra como se ha programado dicha sección.

```
while ( rs.next())
{
vector.add( rs.getString(2) + "A" + rs.getString(3));
}
```

Se puede observar como para pasar de una muestra a la siguiente se usa la función rs.**next**() incluida en un bucle while. Este bucle parará cuando la referencia a la siguiente muestra sea nula.

También se puede apreciar como para la obtención de los datos es necesario la función **getString**(). Los números 2 y 3 hacen referencia a las columnas de la tabla, en este caso los valor 2 y 3 corresponden al valor de la muestra y fecha de obtención respectivamente.

Y por último se devolvería el vector con todas las muestras almacenadas hasta el momento.

Acto seguido se utiliza la clase XYChart.java para poder usar el método **createXYAreaChart**(), para así conseguir representar en un gráfico el historial con todas las muestras, como se puede ver en la siguiente figura:

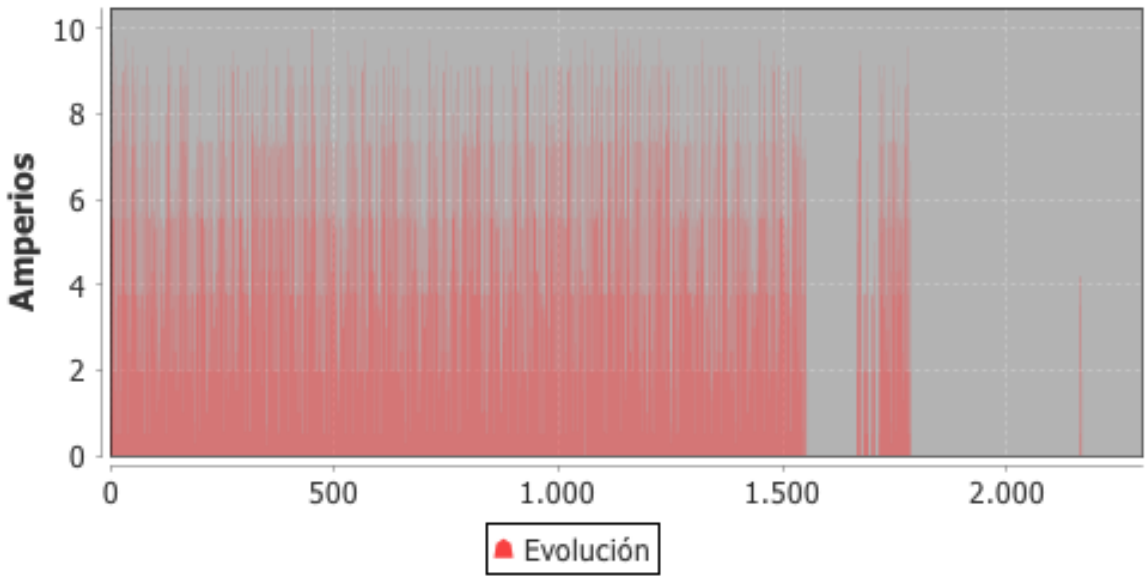

Figura 28: Histórico de muestras en el servidor

- Si se elige la opción "4) Guardar histórico", se archivará un histórico en la pantalla de inicio del ordenador.

La realización de este apartado es muy parecido al anterior, con la diferencia que en vez de usar el método anteriormente comentado, **createXYAreaChart()**, se usará uno para guardar:

saveChartAsJPEG(archivo, figura, tamaño_x, tamaño_y)

En el que el primer parámetro hace referencia a la ruta en donde se guardará el archivo. El segundo parámetro, se trata de los datos de la gráfica y los últimos parámetros corresponden a las dimensiones vertical y horizontal de éste.

Para la realización de este proyecto se ha decido que se guarde el histórico en el escritorio del ordenador.

- "5) Consumo medio en un mes", se procede a mostrar en pantalla el promedio de muestras para un determinado mes.

  Para ello, se vuelve a necesitar usar la base de datos y se realiza una consulta con el mes solicitado por el usuario.

  Dicha consulta, guardará en un vector todas aquellas muestras correspondientes al mes seleccionado. A continuación se procede a realizar el promedio entre todas las muestras y mostrarlo después en pantalla.

  A continuación se puede observar una consulta para el mes de Febrero de 2015.

```
Ha introducido: 5
5) Consumo medio en un mes ...
Introduzca un numero correspondiente al mes
2
Ha introducido: 2

Consumo medio: 4.5227013A
```

Figura 29: Consulta del consumo medio de un mes

- Si la opción elegida es la última, "6) Seleccionar Waspmote", se procederá a mostrar todas las direcciones IP correspondientes a las Waspmotes que se hayan conectado al servidor, para que cuando el usuario elija una opción, sepa a qué Waspmote enviársela.

Cada vez que se selecciona el menú, se muestra la dirección IP activa para enviar las distintas peticiones. Así, por ejemplo, se podría requerir en un cierto momento saber el consumo actual de un frigorífico y justo después que se mostrase el histórico del consumo del microondas. O bien conocer el consumo medio del mes de enero de la calefacción en contra de lo que consumió un determinado mes de verano.

A continuación se muestra una captura de imagen en la que se puede observar tres Waspmotes en la misma red, las cuales han quedado registradas en el servidor. Además se pide al usuario que elija un dispositivo sobre el cual solicitar futuras peticiones.

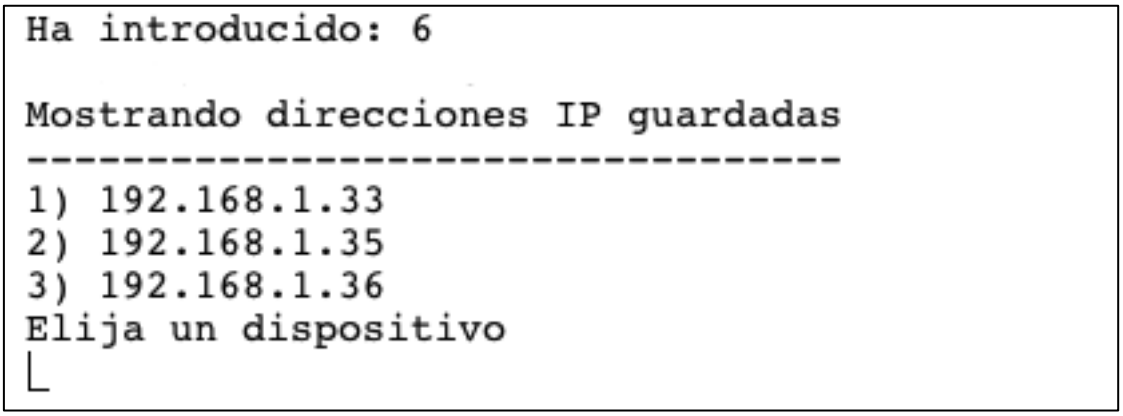

Figura 30: Opción #6 del menú

3.3.3. Aplicación Android

La aplicación en Android es una de las mejoras introducidas en el proyecto para poder conocer en todo momento el consumo actual de un determinado electrodoméstico de forma remota, desde un dispositivo móvil.

Esta parte está basada en el modo de operación cliente-Waspmote ya comentado. La idea consiste en que el terminal móvil solicite unas muestras al servidor, el cual retransmitirá dicha petición a la Waspmote. A continuación la Waspmote tomará la muestra de corriente y se la enviará al servidor que guardará dicha muestra en la base de datos para después reenviársela a la aplicación móvil que la solicitó.

Para esta realización no se ha modificado la parte del servidor, anteriormente descrita sino que se ha añadido una nueva conexión. A continuación se puede el escenario final del proyecto:

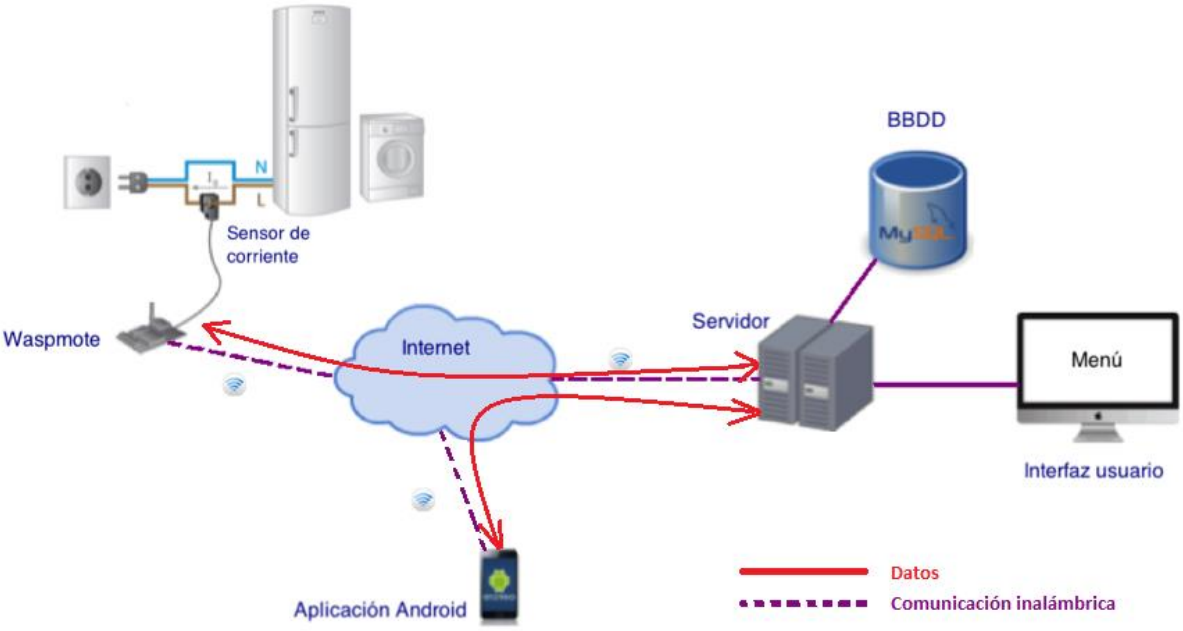

Figura 31: Escenario final del proyecto

La conexión inalámbrica entre el servidor y el dispositivo se ha hecho también con sockets como ocurría con la comunicación Waspmote-servidor, sin embargo para esta parte se ha hecho con sockets TCP, ya que cuenta con las siguientes propiedades:

- Orientados a conexión
- Se garantiza la transmisión de todos los datos sin errores
- Se garantiza que todo llegará en orden.

A continuación se comentará la aplicación móvil y la parte asociada a la misma, que se ha necesitado añadir en servidor.

Dispositivo Móvil

Para la explicación de esta parte se procederá a hablar de las clases y métodos más relevantes para poder llevar a cabo la comunicación con el servidor.

AndroidManifest.xml

Toda aplicación diseñada para dispositivos Android debe contener un archivo AndroidManifest.xml en el directorio raíz. Dicho archivo contiene información esencial acerca de la aplicación para el sistema.

Entre algunas de sus funcionalidades se encuentran las de asignar los permisos que la aplicación debe tener para acceder a partes protegidas del API e interactuar

con otras aplicaciones, declarar el nivel mínimo de la API de Android que requiere la aplicación y describir los componentes de la aplicación.

Además para poder utilizar la red WiFi en el dispositivo Android, se ha tenido que incluir los permisos de acceso a internet:

```
<uses-permission android:name="android.permission.INTERNET" >
```

Figura 32: Permisos WiFi para dispositivos Android

Interfaz

El siguiente paso para la realización de esta aplicación es el diseño de la interfaz gráfica que proporciona al usuario un entorno visual sencillo.

Una característica del sistema Android es ofrecer la posibilidad de crear las interfaces gráficas a través ficheros XML, lo que permite separar la funcionalidad de las clases de la estética. De esta forma, el código queda mucho más ordenado y claro, indicando en los archivos XML la distribución de nuestros objetos gráficos (botones, etiquetas, imágenes...), así como su color, forma, tamaño, colocación etc., mientras que en los archivos java podemos centrarnos en la funcionalidad de éstos y su integración en el sistema.

Estos ficheros se deben almacenar en el directorio /res/layout de nuestro proyecto.

A continuación se va a mostrar el icono de la aplicación elegida para este proyecto:

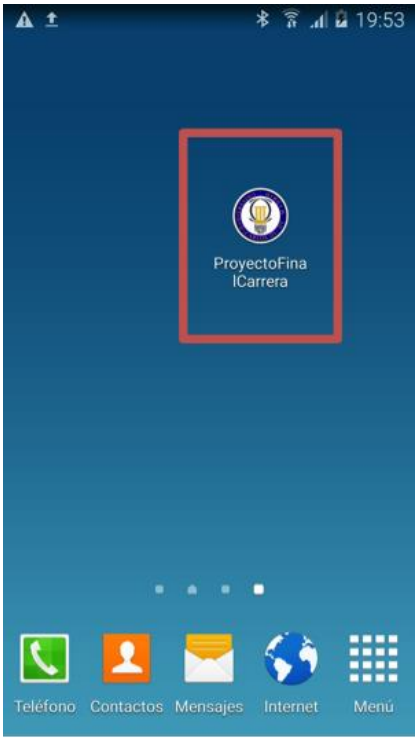

Figura 33: Icono de la aplicación Android

Clases java

La primera clase a comentar es la clase **inicio**.java, que se encarga de manejar la pantalla principal. Dicha clase presenta la aplicación y muestra un botón que cuando se pulse dará acceso a la siguiente pantalla, como se puede observar a continuación.

Figura 34: Página de inicio de la aplicación Android

Para conseguir que cuando se pulse el botón, de acceso a la siguiente pantalla se han necesitado las siguiente líneas de código.

```
Button accionEntrar = (Button) findViewById (R.id.bEntrar);
accionEntrar.setOnClickListener (new View.OnClickListener() {

   public void onClick(View view) {
        Intent intent = new Intent (getApplicationContext (),PFC.class );
        startActivity( intent );
    }
});
```

En dichas líneas se observa como cuando ocurre un nuevo evento, en este caso el pulsar el botón, se da paso a la siguiente clase **PFC**.java y por consiguiente a la siguiente pantalla:

Figura 35: Solicitud muestra en aplicación Android

En esta segunda pantalla de la aplicación se puede observar en la parte superior izquierda, la dirección IP que se ha obtenido al conectarse a la red WiFi.

Cada vez que se quiera conocer el consumo actual de un determinado dispositivo habrá que pulsar el botón naranja "Obtener valor instantáneo".

El modo de creación de esta parte es muy parecido a la comunicación entre el servidor y la Waspmote.

Lo primero que se hace es crear la conexión con el servidor mediante sockets TCP, en el que se define la dirección IP del servidor y el número de puerto.

```
Socket client = new Socket("192.168.1.34", 4444);
```

Nótese que es un puerto distinto al usado entre el servidor y la Waspmote.

Una vez se ha pulsado el botón, el dispositivo Android mandará una petición al servidor solicitándole una muestra. Una vez el servidor la mande de vuelta, ésta será presentada al usuario tal y como muestra la siguiente imagen:

Figura 36: Mostrando resultado final en la aplicación Android

Servidor

En cuanto a la parte del servidor, para poder llevar a cabo la comunicación con el dispositivo Android se ha creado una clase que extiende de Thread, al igual que las anteriores en el servidor, llamada ComunicacionAndroid, para que permita al resto de clases del servidor funcionar sin que se interfieran. Por tanto el diagrama de bloques final resultaría:

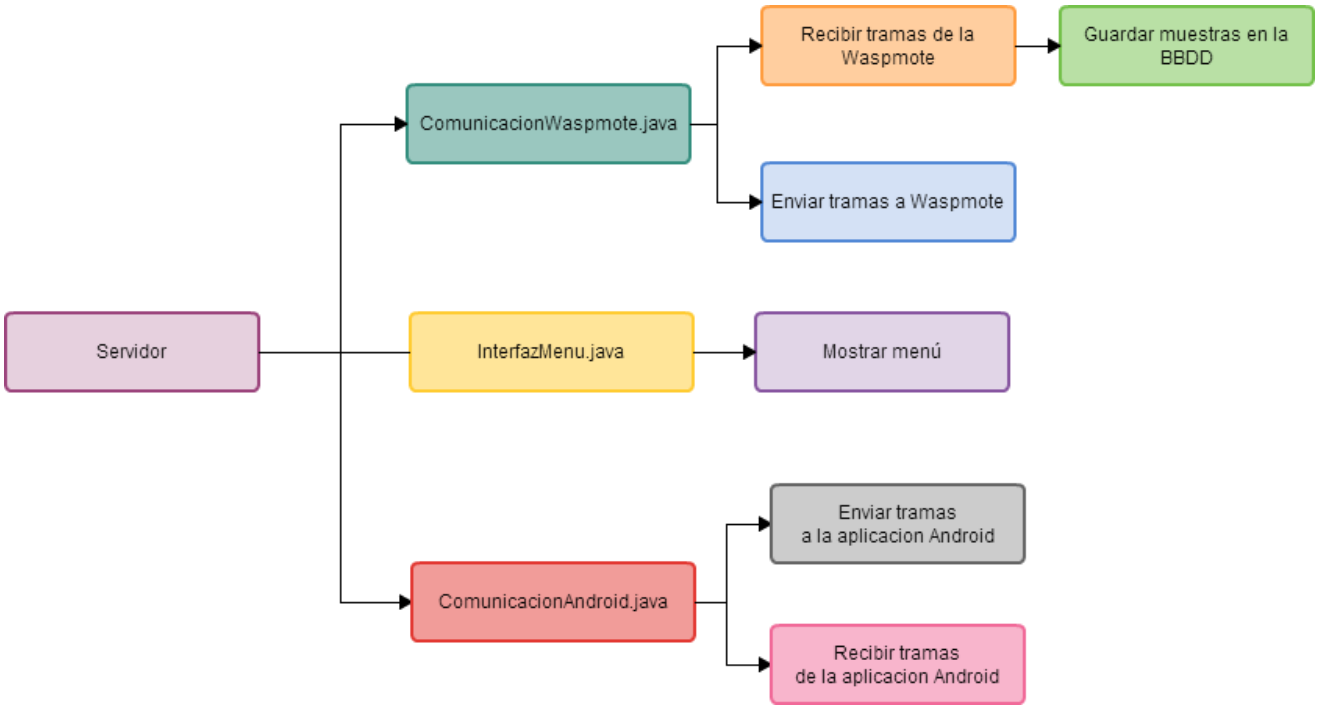

Figura 37: Diagrama de bloques del proyecto final

La clase **ComunicacionAndroid**.java tiene un único método, run(), el cual se repetirá indefinidamente a la espera de que le llegue un mensaje por parte de la aplicación Android.

Para ello se ha creado un socket que siempre está a la escucha en el puerto 4444. Una vez le llega un mensaje proveniente del dispositivo Android, se le notifica a la clase **ComunicacionWaspmote** que envíe un mensaje solicitando una muestra de corriente, como se explicó en la sección del servidor.

Acto después la Waspmote tomará dicha muestra y se la enviará de vuelta al servidor. Y como se comentó, todas las muestras que llegan se guardarán en la BBDD.

Después se procede a notificar a la clase **ComunicacionAndroid**.java dicha recepción y se procede a extraer el último valor guardado de la BBDD, que coincidirá con el anteriormente solicitado y se envía al dispositivo Android.

```
DataOutputStream doS = writeBytes(valorCorriente + "A");
```

A continuación se puede ver el envío de tramas completo que transcurre desde la aplicación Android hasta la Waspmote:

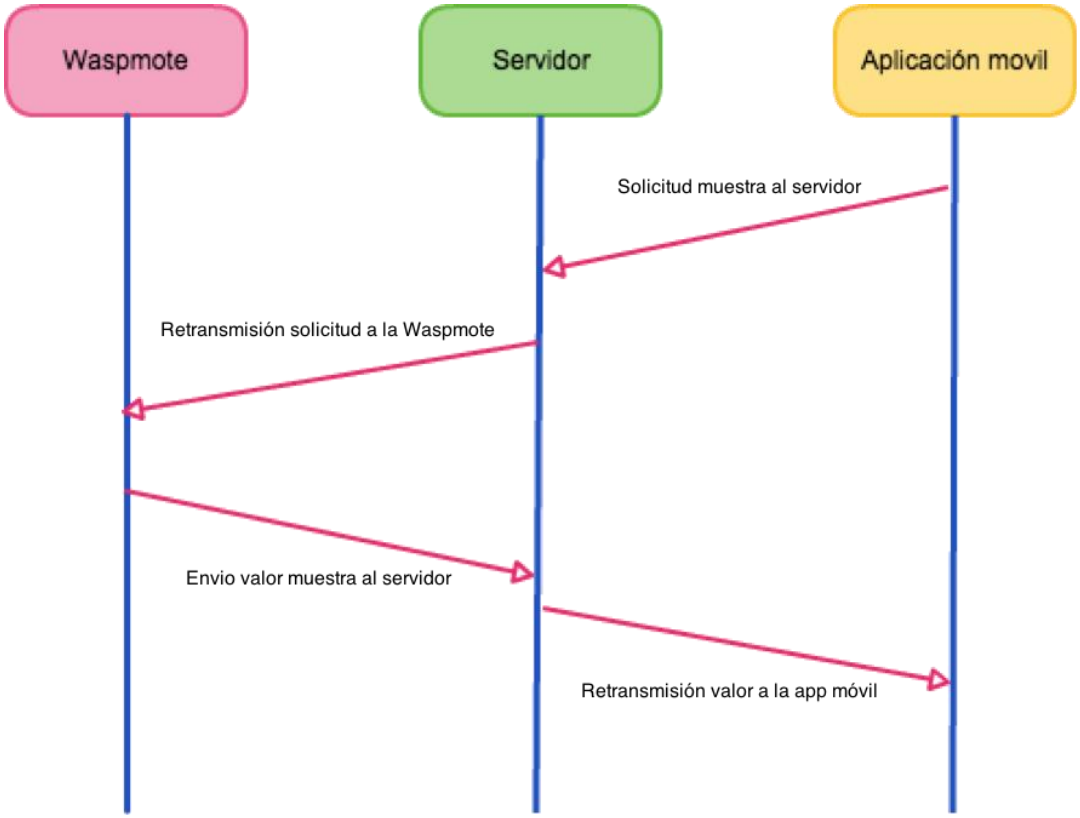

Figura 38: Intercambio de tramas desde la aplicación móvil a la Waspmote

Capítulo 4

Evaluación de la solución

En este capítulo se explican las distintas pruebas a las que se ha sometido el proyecto y que han permitido comprobar el correcto funcionamiento del mismo. Durante el desarrollo de este trabajo se han llevado a cabo comprobaciones en los diferentes módulos para garantizar que están exentos de errores.

4.1. Pruebas

Éstas se han dividido en tres bloques.

4.1.1. Waspmote

Como núcleo principal del proyecto es indispensable que se compruebe la ausencia de errores para que el sistema resulte fiable.

- Sensor de corriente: Comprobación de la toma de corriente con distintos electrodomésticos.

  En la figura 39 se muestra el valor de las muestras de corriente al incrementar la potencia de funcionamiento de un secador de pelo.

  Además se comprobó con un polímetro la validez de estas muestras de corriente.

Figura 39: Muestras de corriente obtenidas al medir un secador

- Módulo WiFi: Conexión al punto de acceso y obtención de los parámetros de red.

  En la siguiente captura de pantalla se puede comprobar la correcta conexión con la red WiFi, mostrándose en la siguiente tabla todos los parámetros relativos a la conexión.

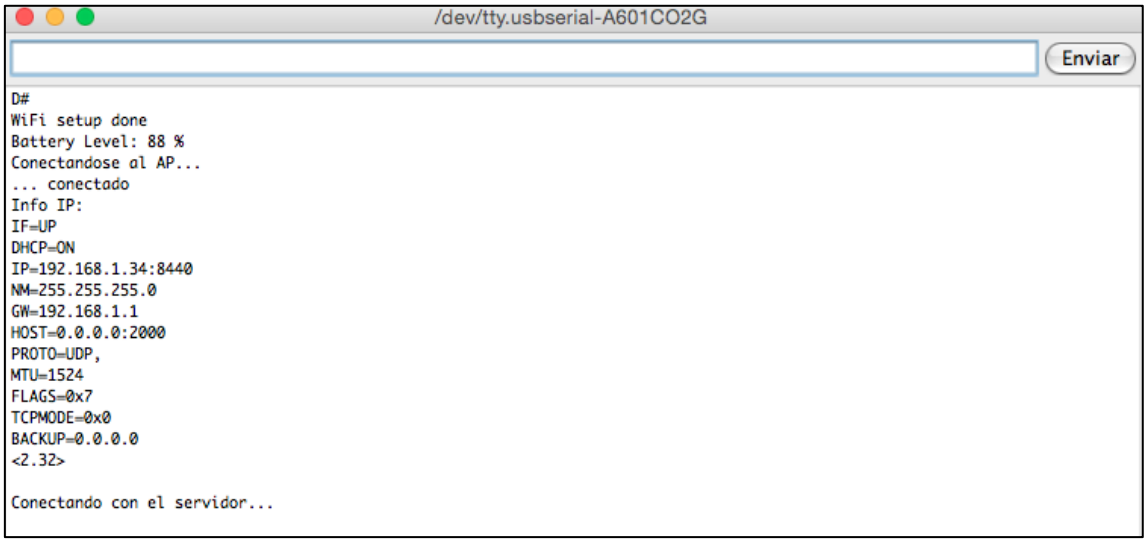

Figura 40: Conexión WiFi de la Waspmote

- Módulo de almacenamiento: Comprobación del almacenamiento y recuperación de muestras de corriente en la memoria SD.

  Se realizaron varias medidas de corriente que se guardaron en la memoria SD. A continuación se extrajeron dichos valores y se comprobó que eran las mismas que se almacenaron.

  A continuación se puede ver en el siguiente ejemplo la extracción de información de la memoria SD.

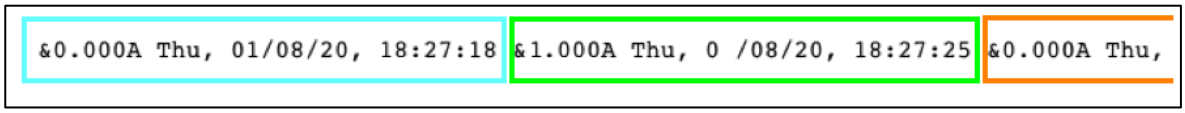

Figura 41: Almacenamiento de muestras en la memoria SD

- Temporización: Necesario para poder llevar una monitorización con el tiempo de muestreo seleccionado por el usuario.

  A continuación se muestra un ejemplo de cómo un temporizador fijado en 5 segundos toma muestras de corriente correctamente espaciadas en el tiempo.

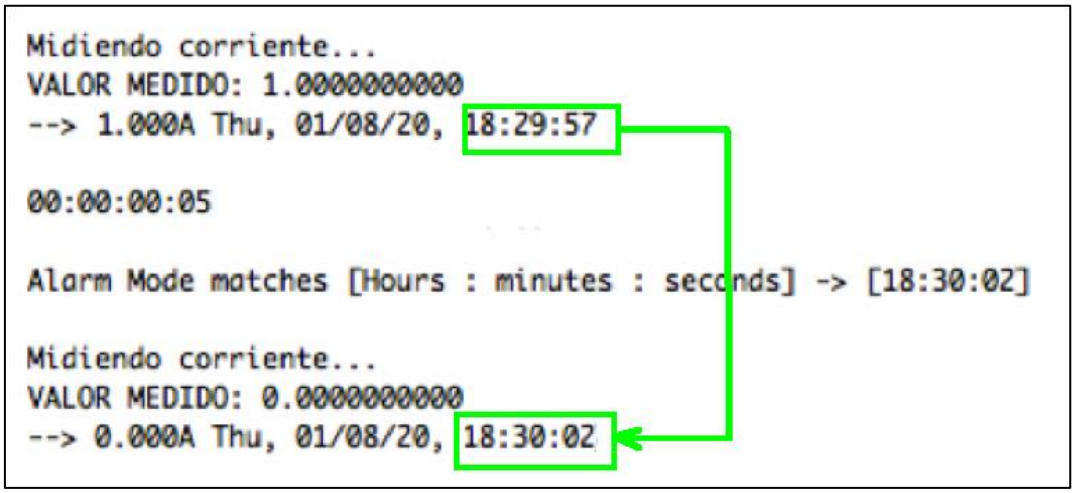

Figura 42: Prueba de temporización en Waspmote

4.1.2. Servidor

Dicha sección juega un papel crucial ya que es donde se guardarán todas las muestras y además se controla las conexiones entre dispositivos. A continuación se muestra el escenario de pruebas usado.

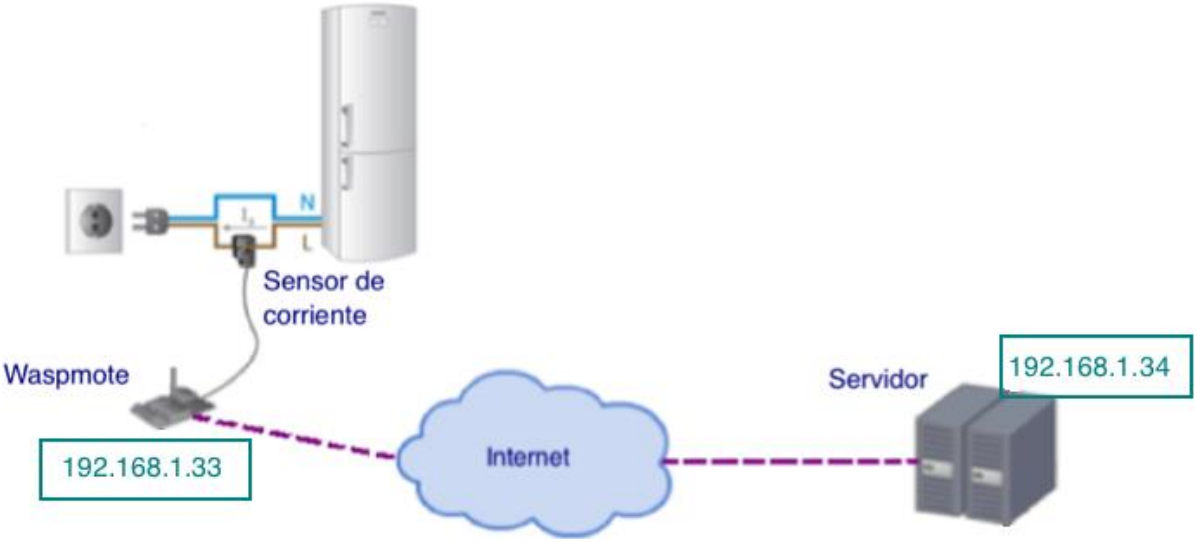

Figura 43: Escenario prueba Servidor-Waspmote

- Comunicaciones: Comprobación de la recepción y envío de tramas.

  En dicho escenario de pruebas se enviaron distintos mensajes desde la Waspmote al servidor y se comprobó que se recibían correctamente. También se realizó a la inversa, es decir, se mandaron mensajes desde el servidor y se comprobó que llegaban correctamente en la Waspmote.

  A continuación se muestra el establecimiento de la conexión entre ambos, en el que la Waspmote solicita dicho establecimiento al servidor y éste le responde con la hora y fecha actuales:

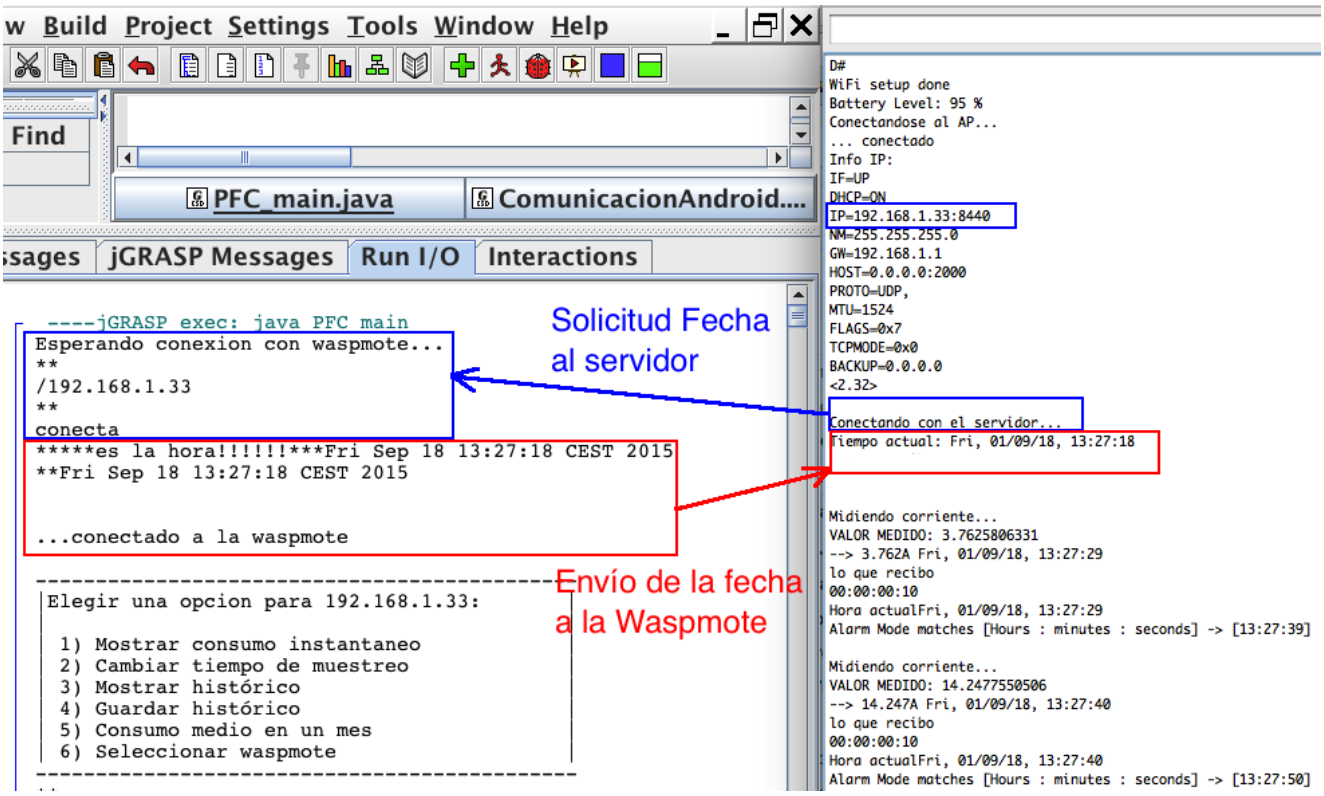

Figura 44: Establecimiento de la comunicación entre la Waspmote y el servidor

Del mismo modo se puede comprobar la solicitud de una muestra de corriente desde el menú en la siguiente imagen:

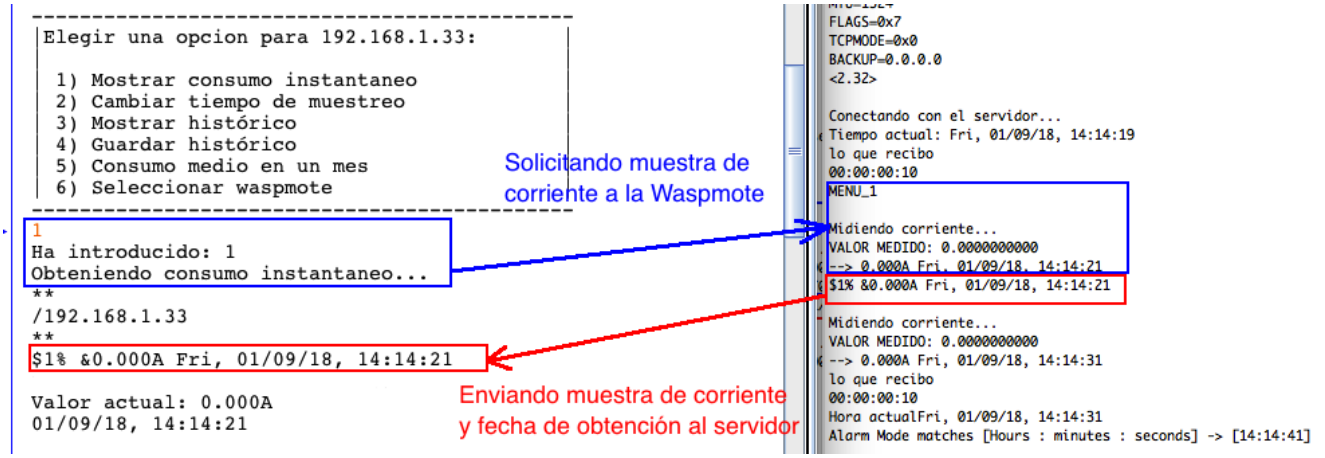

Figura 45: Intercambio de mensajes al solicitar una muestras de corriente

- Cambio del tiempo de las tomas de corriente

  Se puede comprobar el correcto comportamiento del sistema al solicitar un cambio en el tiempo de adquisición de las muestras. Así como comprobar que dicho cambio funciona, al tomar la siguiente muestra de corriente pasados 20 segundos.

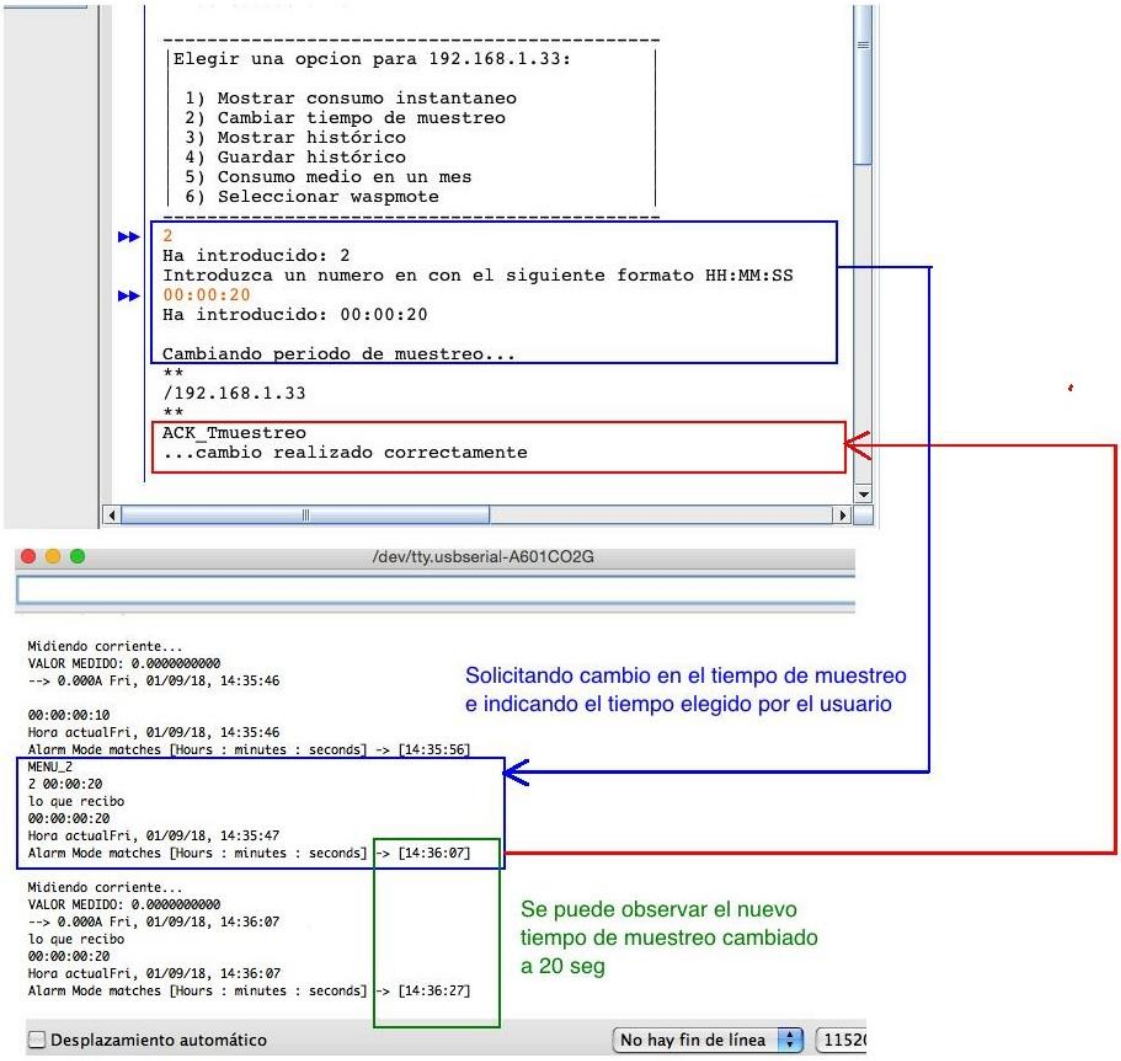

Figura 46: Intercambio de mensajes al cambiar el tiempo de muestreo

- Base de datos: Comprobación del correcto almacenamiento de muestras en la BBDD. Para ello se comprobó que las muestras que habían sido tomadas por la Waspmote eran las mismas que se guardaban en la BBDD.

  A continuación se puede apreciar el almacenamiento de distintas muestras tomadas por el sensor de corriente. También se puede apreciar en la siguiente figura cómo las muestras de corriente son presentadas en la pantalla que refleja el histórico.

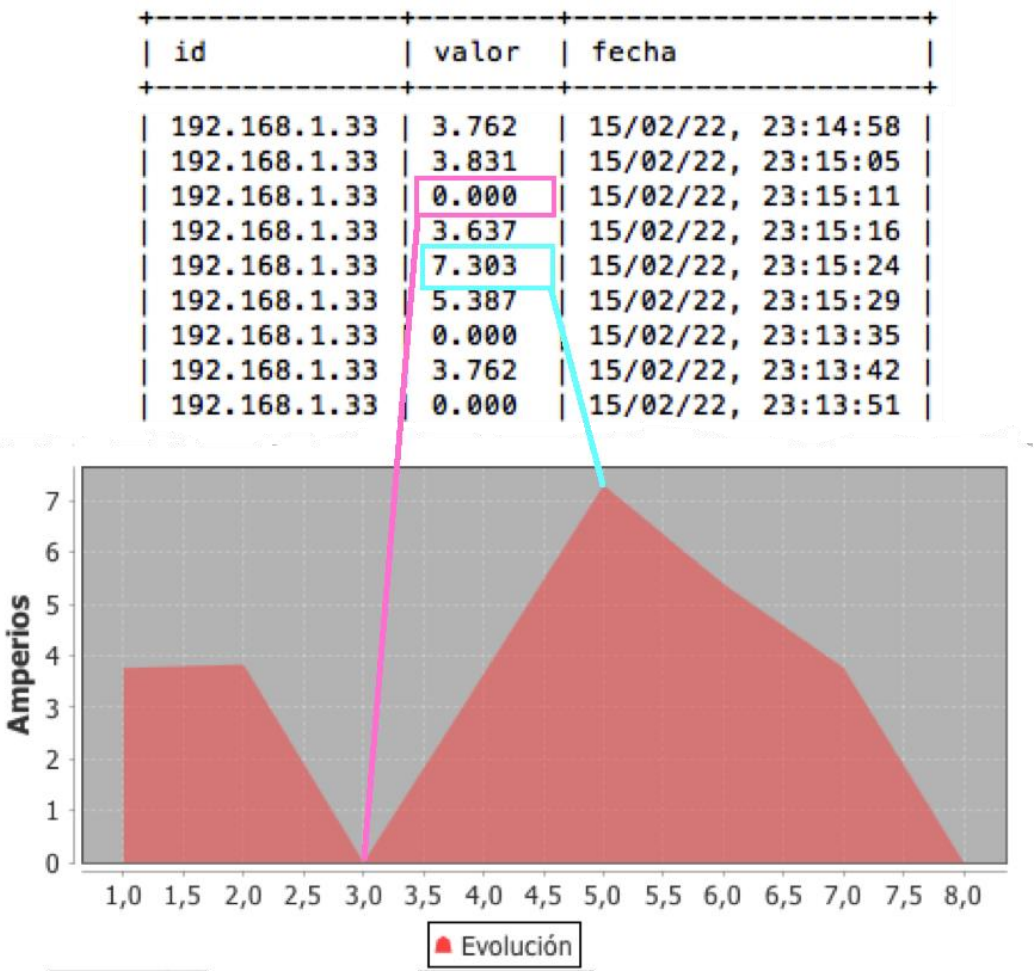

Figura 47: Ejemplos de muestras guardadas en la BBDD y de su histórico

También se puede comprobar en la siguiente imagen, seleccionando el modo 5, el consumo medio para el mes de febrero.

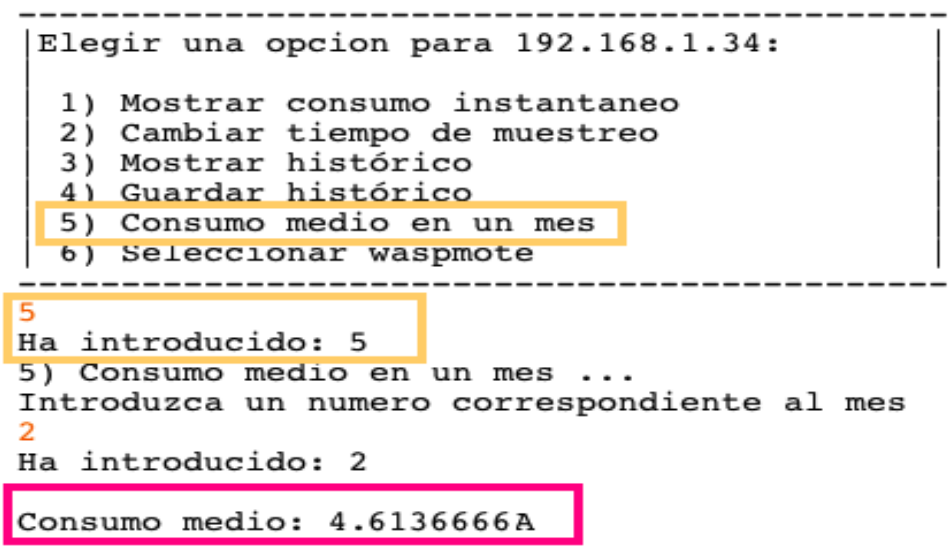

Figura 48: Ejemplo de consulta del consumo medio del mes de febrero

- Almacenamiento correcto de distintas Waspmotes en el servidor, para poder seleccionar adecuadamente la Waspmote a la que realizar una petición.

  A continuación se puede observar el almacenamiento de muestras de corriente correspondientes a diferentes Waspmotes en un escenario con dos Waspmotes.

id	valor	fecha
192.168.1.33	0.000	15/02/22, 20:45:33
192.168.1.33	0.000	15/02/22, 20:45:37
192.168.1.33	0.000	15/02/22, 20:45:40
192.168.1.33	5.387	15/02/22, 21:39:09
192.168.1.34	7.423	15/02/22, 21:39:20
192.168.1.34	5.268	15/02/22, 21:39:28
192.168.1.33	0.000	15/02/22, 21:39:37
192.168.1.33	10.000	15/02/22, 23:06:57
192.168.1.33	8.000	15/02/22, 23:07:02
192.168.1.34	9.000	15/02/22, 23:07:10
192.168.1.33	10.000	15/02/22, 23:07:40
192.168.1.33	0.000	15/02/22, 23:10:21

Figura 49: Almacenamiento muestras de distintas Waspmotes

4.1.3. Aplicación Android

El escenario elegido para realizar las pruebas con el dispositivo Android es el siguiente:

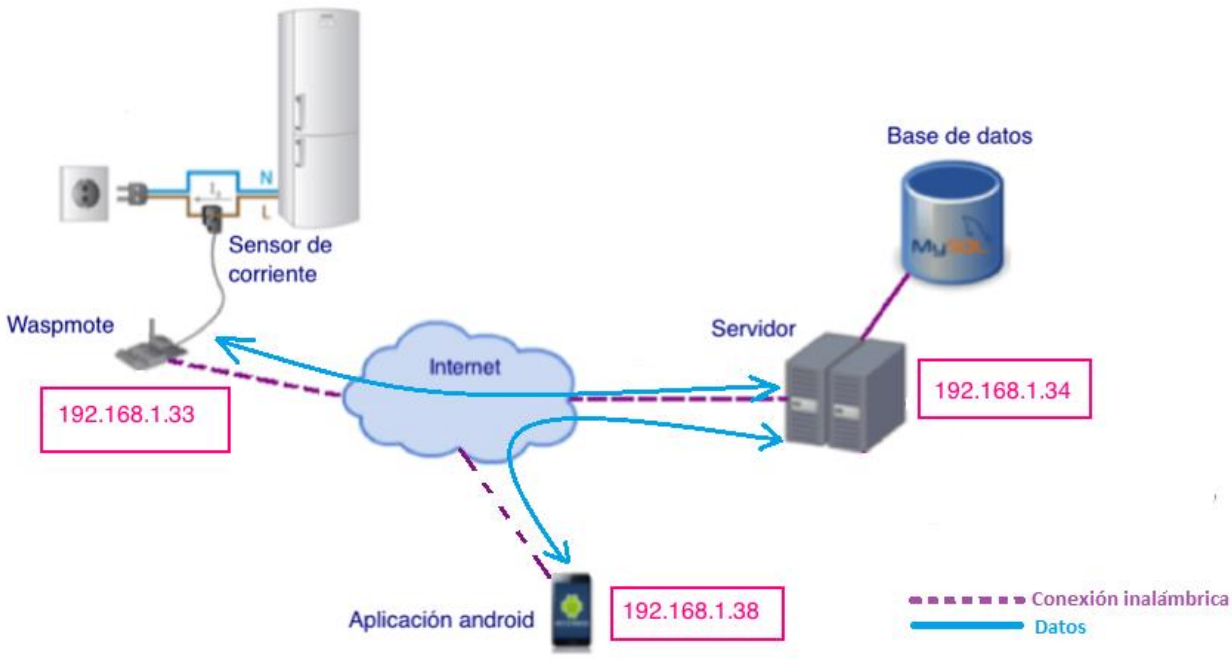

Figura 50: Escenario de pruebas final

- Prueba final con el que se comprueba el correcto funcionamiento de la aplicación.

  Una vez se pulsa el botón de la aplicación para solicitar la medida del consumo, se envían los correspondientes mensajes al servidor y a la Waspmote.

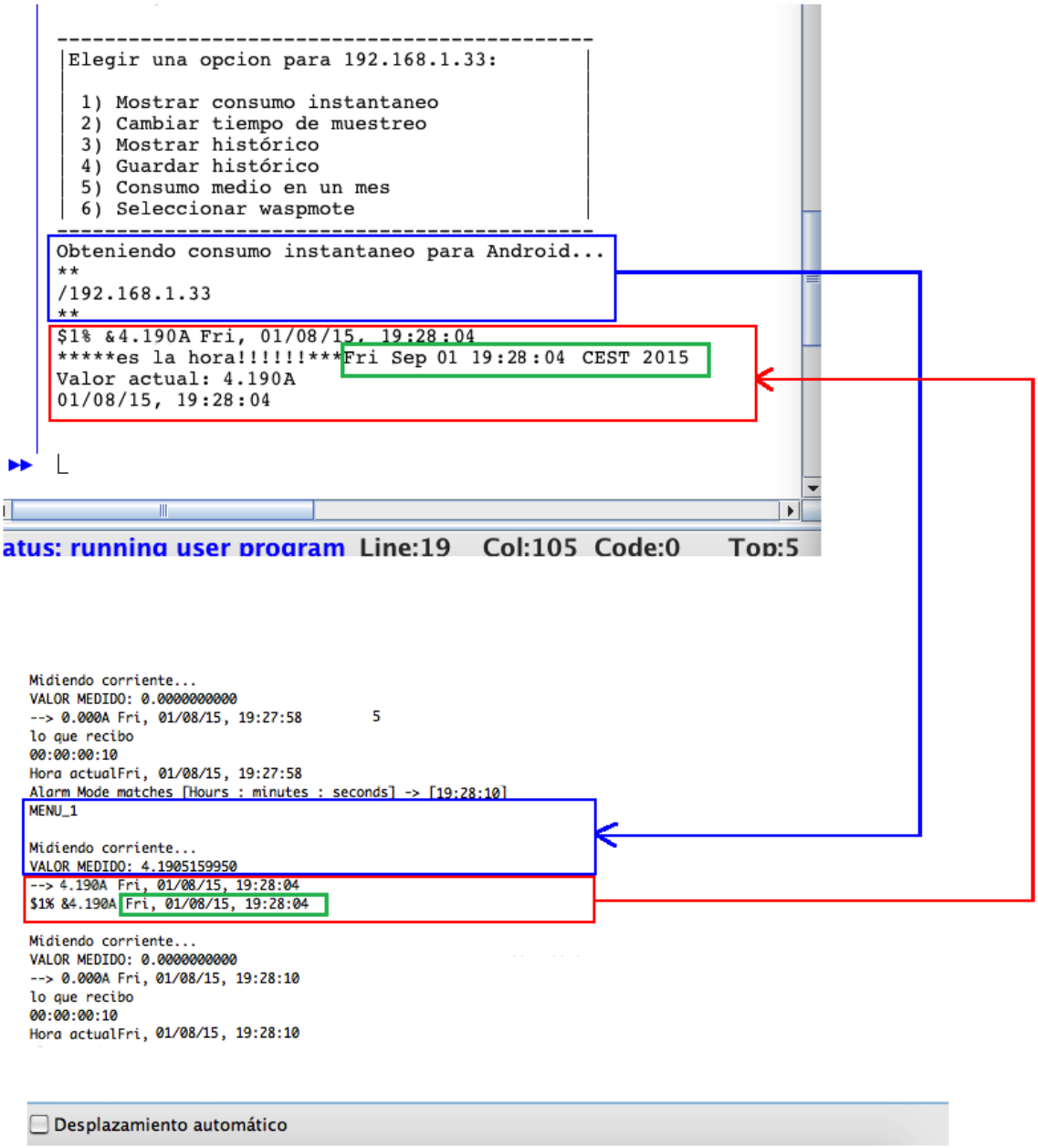

Figura 51: Solicitud muestra desde dispositivo Android

Se puede apreciar, resaltado en verde, cómo el tiempo y fecha de obtención de la muestra se envía correctamente.

Tras la recepción del valor de la muestra de corriente ésta se guarda en la BBDD y se envía al dispositivo Android que lo solicitó por pantalla.

En la siguiente imagen se puede observar como el valor y fecha almacenada en la BBDD corresponde con la que se muestra en el móvil.

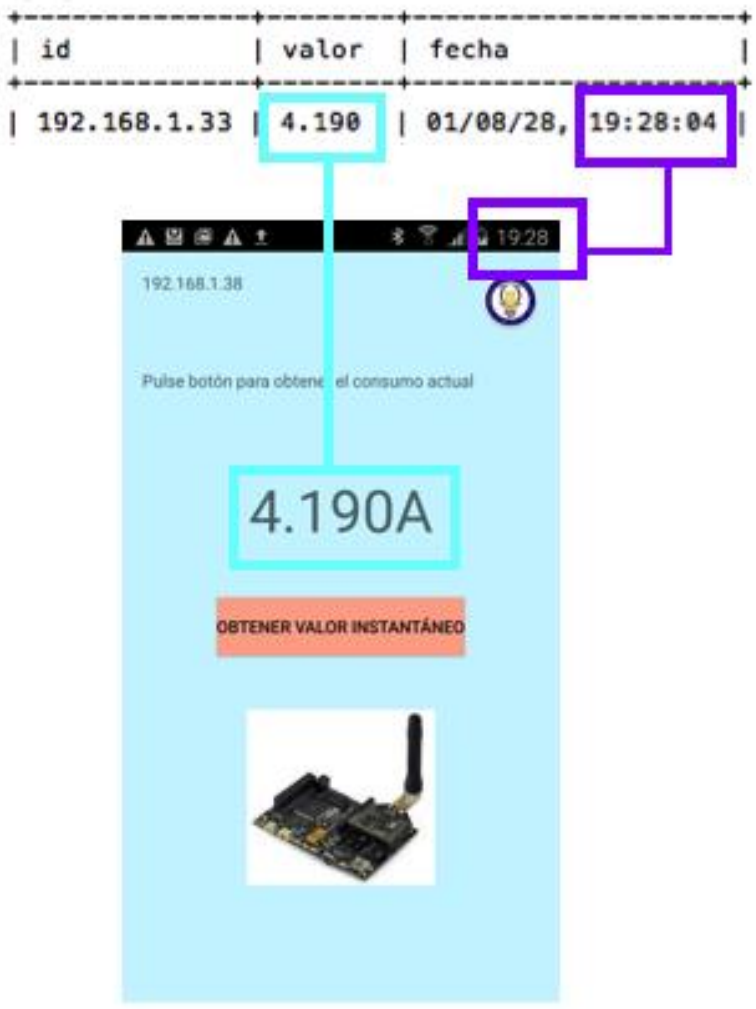

Figura 52: Valor obtenido en dispositivo Android

4.2. Consumos

En esta sección se pretende mostrar la duración que tiene la batería de la Waspmote tanto de manera general como en particular para este proyecto.

Primeramente hay que comentar los dos modos de funcionamiento a la hora de realizar dichas medidas.

El primero trata de poner a la Waspmote en modo hibernación. Esto significa que la Waspmote no realizará ninguna función mientras se encuentre en este estado. Dicha inactividad durará hasta que venza un temporizador.

Y el segundo se trata del estado de no hibernación. Dicho estado es el utilizado en este proyecto debido, como ya se comentó, a que si se dormía la Waspmote no podría recibir ninguna trama por parte de servidor vía WiFi, debido a las características de ésta.

El primer resultado a comentar consiste en las dos siguientes pruebas. Estas consisten en ejecutar la Waspmote para que envíe al servidor el porcentaje de batería actual cada minuto tanto para el caso en que se utiliza hibernación como para el que no se utiliza.

A continuación se muestran los resultados obtenidos:

Hibernación	Frecuencia de envío WiFi	Duración (DD:HH:MM)
NO	1 minuto	04:23:09
SI	1 minuto	30:12:17

En donde se puede ver que al no utilizar hibernación en la Waspmote hace que el tiempo sea bastante menor. Esto quiere decir que el ejecutar la Waspmote con esperas activas, éstas consumen muchos recursos, como cabía esperar.

La siguiente prueba llevada a cabo fue el incrementar la frecuencia de envío del porcentaje actual de la batería al servidor para el caso de no hibernación y ver como influía.

Hibernación	Frecuencia de envío WiFi	Duración (DD:HH:MM)
NO	5 minuto	05:03:50
NO	60 minutos	06:14:26

Se puede observar que a mayor frecuencia de envío de información al servidor, la Waspmote consumirá más.

Por otro lado se realizó una última prueba en modo hibernación:

Hibernación	Frecuencia de envío WiFi	Duración (DD:HH:MM)
SI	60 minuto	49:23:12

Se puede corroborar como en el modo de hibernación la duración de la batería se alarga.

Por otro lado se puede observar como también influye notablemente la frecuencia con la que se realiza la comunicación WiFi, ya que a medida que se tarde más en enviar un mensaje al servidor, la vida de la batería se alarga considerablemente.

A continuación se realizarón las siguientes pruebas usando el código desarrollado para este proyecto.

En estas, cada minuto se toma una muestra de corriente que se guarda en la memoria SD y se envía al servidor con distintas frecuencias.

Hibernación	Toma de muestra de corriente	Frecuencia de envío WiFi	Duración (DD:HH:MM)
NO	1 minuto	1 minuto	03:23:12
NO	1 minuto	60 minutos	05:06:41

Se puede observar como la frecuencia de envío influye notablemente en la duración de la batería de la Waspmote, como se había anticipado anteriormente.

Por ultimo comentar que se ha podido apreciar que cuando el nivel de batería se reduce, por debajo de un 15%, la Waspmote empieza a tener problemas de conexión a la red WiFi.

Capítulo 5

Conclusiones y líneas futuras

Tras haber concluido el desarrollo de este Proyecto Fin de Carrera, es el momento de efectuar el análisis y balance del resultado final obtenido, así como de futuras extensiones que se le puedan dar al mismo.

5.1. Conclusiones

Con la realización de este proyecto fin de carrera se ha logrado implementar un sistema que monitoriza el consumo de electricidad de un dispositivo conectado a una toma de corriente eléctrica.

Ante la actual implantación de contadores de luz, que miden el consumo por periodos de tiempo, y ante las nuevas tarifas que varían en función del tramo horario, este proyecto permite facilitar la información necesaria para revisar los hábitos de consumo eléctrico y permitir en función de esto tomar las medidas necesarias para reducir la factura de la electricidad.

Así mismo se ha realizado una aplicación en Android que permite monitorizar los consumos de forma remota y sin necesidad de tener que estar delante de la pantalla del ordenador.

Para la realización de este proyecto se ha elegido una plataforma hardware open source de bajo coste. No obstante, después de realizar este proyecto resultaría factible el trasladar la idea desarrollada a otras plataformas tales como Arduino o Rasberry PI.

El mecanismo de conexión elegido ha sido mediante WiFi, no obstante sería posible utilizar como se ha comentado en apartados anteriores otros procedimientos de conexión tales como ZigBee, Bluetooth o incluso 3G, teniendo

en cuenta las limitaciones de cada una de ellas. Del mismo modo, los protocolos elegidos para realizar las conexiones entre los diferentes elementos han sido mediante UDP y TCP.

La base de datos seleccionada ha sido una de código abierto, MySQL, no obstante al ser un módulo independiente del programa principal, podría haberse utilizado cualquier otra base de datos como PostgreSQL, MSSQL, SQLite, MS Access, etc.

La aplicación desarrollada es también fácilmente trasladable a otros sistemas operativos para dispositivos móviles como IOS o Windows Phone.

Realizándose las distintas pruebas a la Waspmote se ha observado que al no utilizar el modo de hibernación se reduce bastante el tiempo de vida de la batería, al igual que el utilizar reiteradas veces el módulo de comunicaciones WiFi.

5.2. Líneas futuras

En esta sección se presentan algunas ideas sobre posibles líneas de desarrollo futuras para continuar con el estudio llevado a cabo en el presente trabajo.

Se podrían añadir muchísimas más funcionalidades al menú para distintos usos, gracias a todas las muestras guardadas en la base de datos.

A continuación se presentan diversas propuestas:

- Comparación de consumos entre distintos dispositivos, para conocer los que más gastan.

- Detectar aquellos meses en los que un determinado electrodoméstico consumió más, por ejemplo aire acondicionado o calefacción.

- El sistema podría recomendar la posibilidad de apagar un determinado electrodoméstico en el caso de que esté consumiendo más energía de la necesaria.

  Para ello se podría mandar un mensaje al usuario en el propio servidor o bien a través de la propia aplicación móvil creada para conocer dicho suceso.

- Almacenamiento del consumo eléctrico mensual y de esta manera se podría comprobar si se corresponde con el mismo importe que el de nuestra compañía eléctrica.

Acrónimos

IP:	Internet Protocol
UDP:	User Datagram Protocol
TCP:	Transmission Control Protocol
ICMP:	Internet Control Message Protocol
AP:	Access Point
NAT:	Network Address Protocol
WiFi:	Wireless Fidelity
WPA:	WiFi Protected Access
RTC:	Real Time Clock
EEPROM:	Electrically Erasable Programmable Read-Only Memory
SD:	Secure Digital
SPI:	Serial Peripheral Interface
WPAN:	Wireless Personal Area Network
FAT16:	File Allocation Table
WAN:	Wide Area Network
WPAN:	Wireless Personal Area Network

Bibliografía

[1] Boungju Jeon, Byeongkwan Kang; Sehyun Park, "Modeling of Electronic Appliance Usage Pattern and Implementation of User Centric Flexible Energy Management System applying Adaptive Energy Saving Policy", in Wireless Information Technology and Sytems (ICWITS), 2012 IEEE International Conference on, vol, no, pp.1-4, 11-16 Nov. 2014
(Última consulta: Nov-2014)

[2] Jinsoo Han; Haeryong Lee; Kwang-Roh Park, "Remote - Controllable and Energy - Saving Room Architecture based on ZigBee Communication" in Consumer Electronics, IEEE Transactions on, vol 55, no.1, pp.264-268, February 2009
(Última consulta: Nov-2014)

[3] In-Ho Choi, Joung-Han Lee; Seung-Ho Park, "Implementation and Evaluation of the Apparatus for Intelligent Energy Management to Apply to the Smart Grid at Home", in Instrumentation and Measurement Technology Conference (I2MTC), 2011 IEEE, vol. no., pp. 1-5, 10-12 May 2011
(Última consulta: Nov-2014)

[4] Starsinic, M., "System Architecture Challenges in the Home M2M Network" in Applications and Technology Conference (LISAT), 2010 Long Island Systems, vol., no., pp. 1-7, 7-7 May 2010
(Última consulta: Nov-2014)

[5] Sensores en el ayuntamiento de Marbella.
http://www.laopiniondemalaga.es/marbella/2012/08/03/marbella-instala-sensores-controlar-consumo-electrico-sedes-publicas/524213.html
(Última consulta: Nov-2014)

[6] ZigBee vs Bluetooth.
https://borealtech.wordpress.com/2006/12/20/bluetooth-vs-zigbee/
(Última consulta: Nov-2014)

[7] Comparativa ZigBee, Bluetooth y WiFi.
http://xbee.wikispaces.com/Conclusion
http://webdelcire.com/wordpress/archives/1714 (Última consulta: Nov-2014)

[8] Sensor de corriente Efergy. http://www.efergy.com/
(Última consulta: Nov-2014)

[9] Sensor de corriente Belkin. http://www.belkin.es/
(Última consulta: Nov-2014)

[10] Sensor de corriente Ciensol. http://www.ciensol.es/
(Última consulta: Nov-2014)

[11] Arduino. https://www.arduino.cc/ (Última consulta: Jul-2015)

[12] Base de datos MySQL https://www.mysql.com/ (Última consulta: Jul-2015)

[13] iOS. http://www.apple.com/es/ios/developer/ (Última consulta: Jul-2015)

[14] Android. https://es.wikipedia.org/wiki/Android (Última consulta: Jul-2015)

[15] Windows Mobile. https://www.windowsphone.com/es-es/features
(Última consulta: Jul-2015)

[16] Documentación de la Waspmote y de todos sus módulos.
http://www.libelium.com/es/development/waspmote/documentation/
(Última consulta: Jul-2015)

[17] Android Studio.
https://developer.android.com/sdk/index.html#Requirements
(Última consulta: Mar-2015)

[18] UDP vs TCP. http://es.diffen.com/tecnologia/TCP-vs-UDP
(Última consulta: Jul-2015)

[19] Precio Waspmote.
https://www.cooking-hacks.com/shop/waspmote/wireless
(Última consulta: Jul-2015)

En esta sección se incluyen los manuales de instalación de los programas, el manual de usuario para poder ejecutar el proyecto, la planificación del proyecto tanto en forma de tareas como temporal y el presupuesto total que ha supuesto la realización de este proyecto.

Anexo I: Manual de usuario

Este apartado sirve para dar a conocer como poder poner en marcha este proyecto.

Waspmote

A continuación se enunciarán todos los pasos que deben ser llevados a cabo para poder hacer uso de la Waspmote.

A. Configuración Hardware

1. Conectar la batería a la Waspmote del mismo modo que se muestra en la figura 53. Recordad que la batería debe estar cargada un mínimo de 24 horas antes de poder usar la Waspmote.

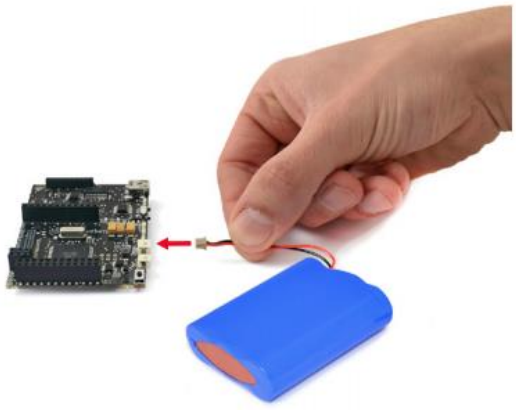

Figura 53: Conexión de la batería a la Waspmote

2. Conectar la antena WiFi en el socket0 de la Waspmote como se muestra en la siguiente figura:

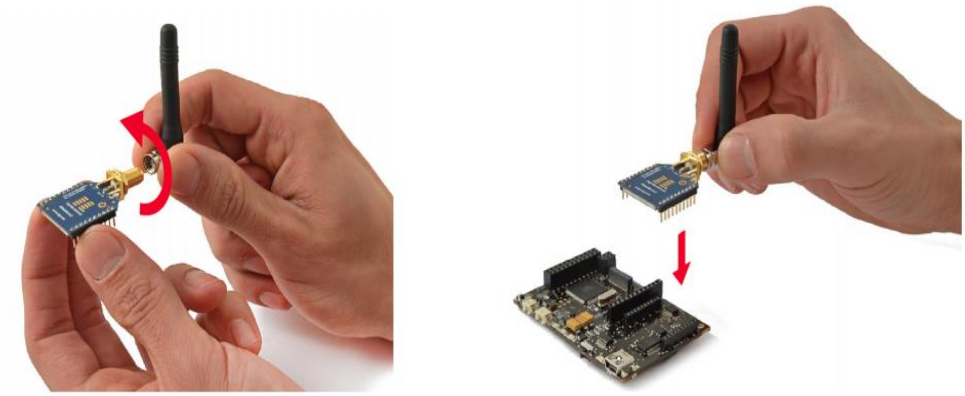

Figura 54: Instalación antena WiFi en la Waspmote

3. Conectar el módulo Smart metering en el conector de la parte superior de la Waspmote, como se muestra en la siguiente figura. Dicho módulo es necesario para poder conectar el sensor de corriente.

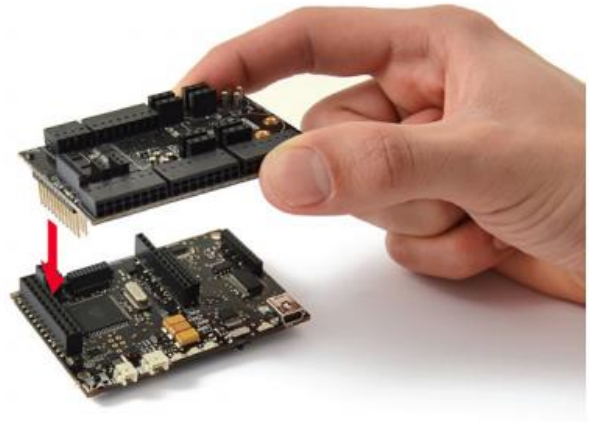

Figura 55: Colocación del Smart metering encima de la Waspmote

4. Conectar el sensor de corriente al socket 1 del Smart metering de la siguiente manera:

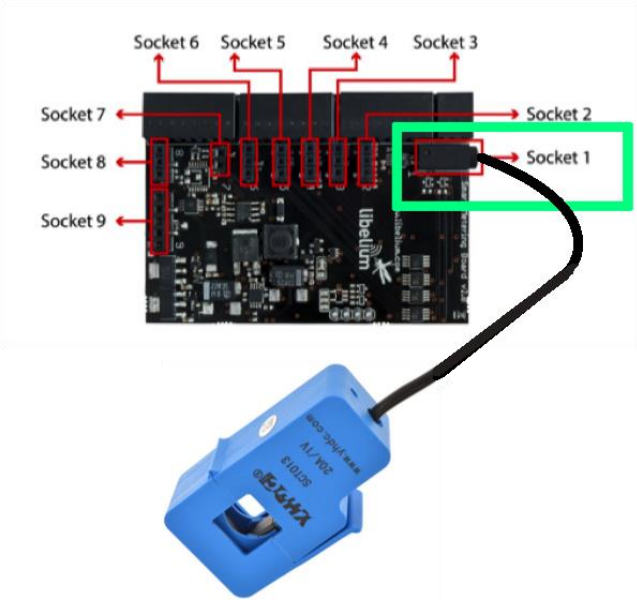

Figura 56: Socket en el que conectar el sensor de corriente

5. Enganchar el sensor de corriente al dispositivo del que se quiera conocer el consumo de la siguiente manera:

Figura 57: Modo de enganchar el sensor en un electrodoméstico

Señalar que el sensor de corriente se debe conectar al cable neutro, y no al cable de fase o al cable entero.

B. Instalación del Software

El siguiente paso consiste en instalar el software Waspmote disponible en el ordenador, para poder cargar desde este el código del proyecto en la Waspmote:

http://www.libelium.com/development/waspmote/sdk_applications/

Válido para entornos Windows, Mac OS y Linux.

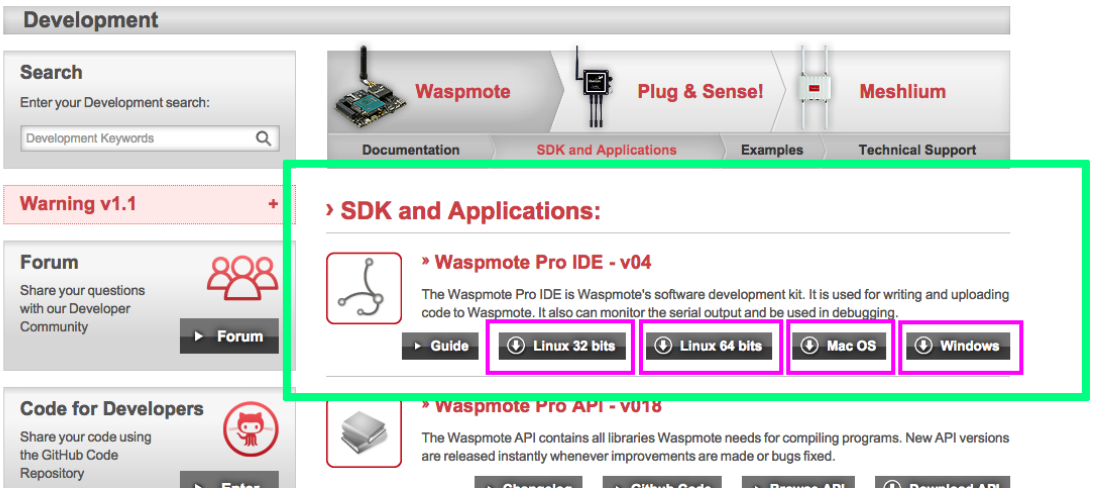

Figura 58: Descarga de la aplicación Waspmote Pro IDE

A continuación se conectará la Waspmote al ordenador mediante un cable USB.

Además se debe de poner la Waspmote en modo ON.

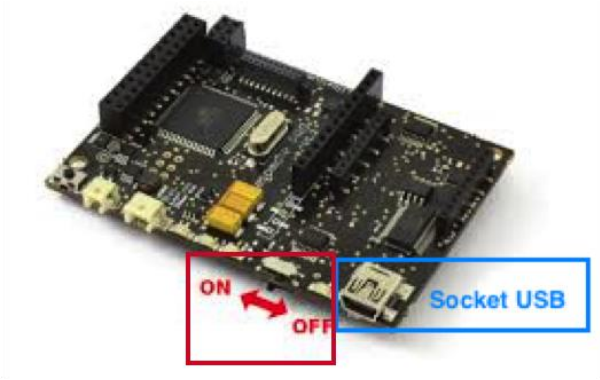

Figura 59: Activación de la Waspmote

C. Descarga del código

El siguiente paso consiste en descargar el código del proyecto y ejecutarlo con la aplicación Waspmote anteriormente descargada.

1. Se debe de modificar el código para utilizar la Waspmote en nuestra red WiFi. Para ello se debe escribir la red y contraseña WiFi a usar, como se muestra en la siguiente figura.

   Además se debe seleccionar la dirección IP de nuestro servidor.

Figura 60: Ejemplo de cambio de la red WiFi y de la dirección IP del servidor

2. Una vez modificado el código se necesita compilarlo pulsado el siguiente botón de la aplicación, resaltado en verde.

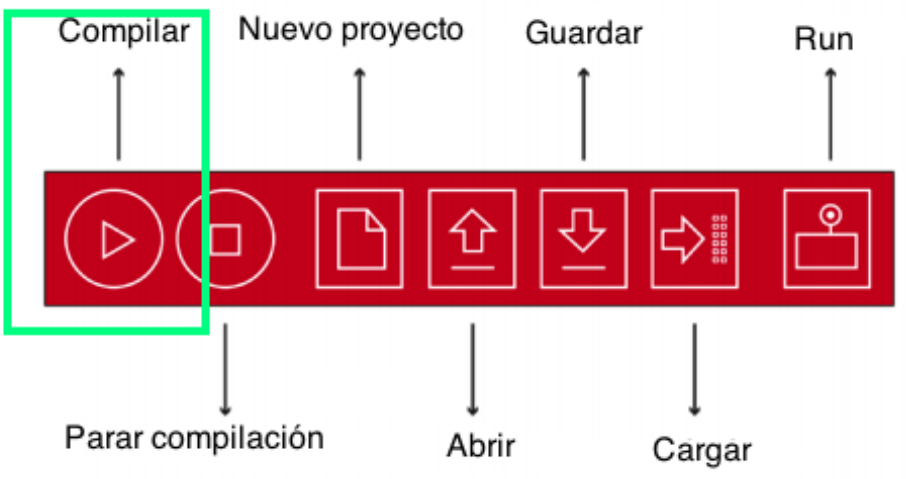

Figura 61: Botón para compilar en la aplicación Waspmote

Una vez finalizado se mostrará un mensaje como el siguiente, en la parte inferior, indicando que la compilación se ha realizado correctamente:

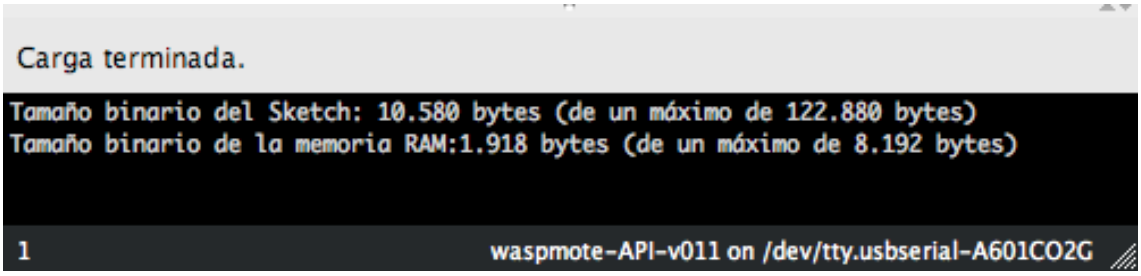

Figura 62: Compilación realizada correctamente

3. Realizados todos estos pasos, la Waspmote comenzará a funcionar, pudiendo proceder a desconectar el cable USB.

   Se puede reiniciar cuando se desee pulsando el botón de reset.

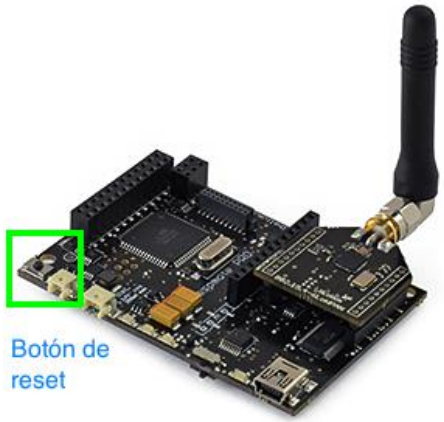

Figura 63: Botón de reset

4. Téngase en cuenta que aunque la Waspmote haya arrancado no empezará a tomar muestras hasta que no se establezca una conexión con el servidor.

Servidor

En esta sección se comentarán los pasos necesarios para poder crear y ejecutar la aplicación del servidor.

Dicha parte se divide en dos secciones. Una primera basada en la creación de la base de datos, y una segunda relacionada con la aplicación propiamente dicha.

A. Instalación de la base de datos MySQL

1. El primer paso consiste en descargar e instalar la base de datos MySQL disponible en:

   http://dev.mysql.com/downloads/mysql/

Figura 64: Descarga de MySQL

2. Una vez instalado es preciso activarlo. Para ello hay que dirigirse a las preferencia del sistema y seleccionar dicha aplicación:

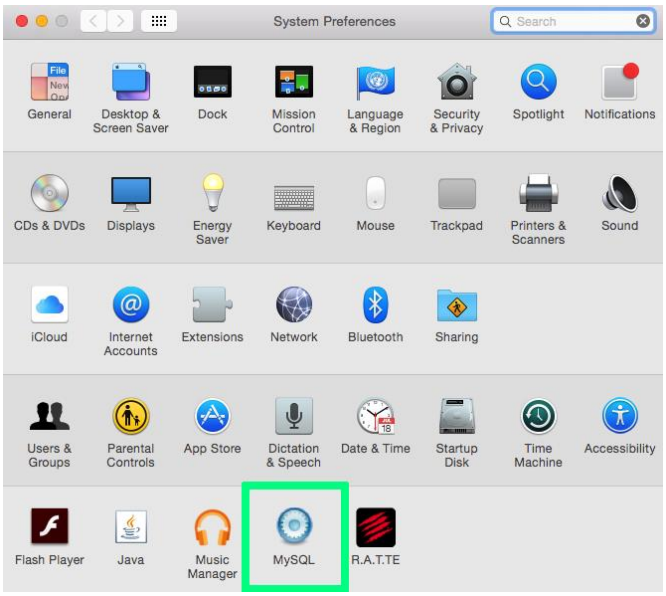

Figura 65: Aplicación MySQL

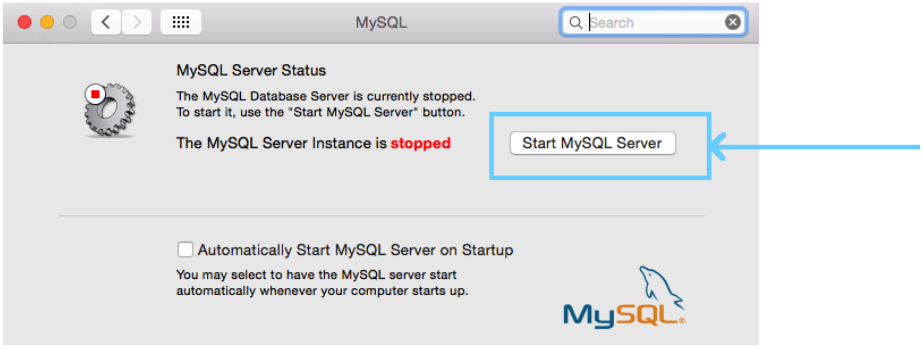

Figura 66: Servidor MySQL desactivado

Y comprobar que está activado:

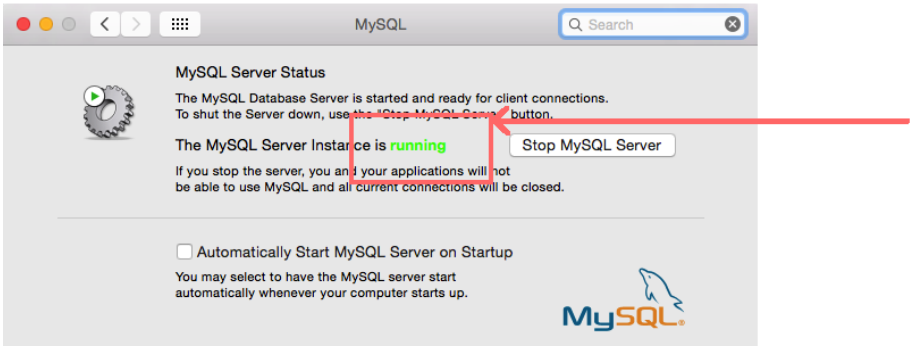

Figura 67: Activación de MySQL

3. El siguiente paso consiste en la creación de nuestra base de datos, en la que se guardarán todas las muestras tomadas.

4. Para ello es necesario abrir un terminal y escribir las siguientes sentencias remarcadas a continuación:

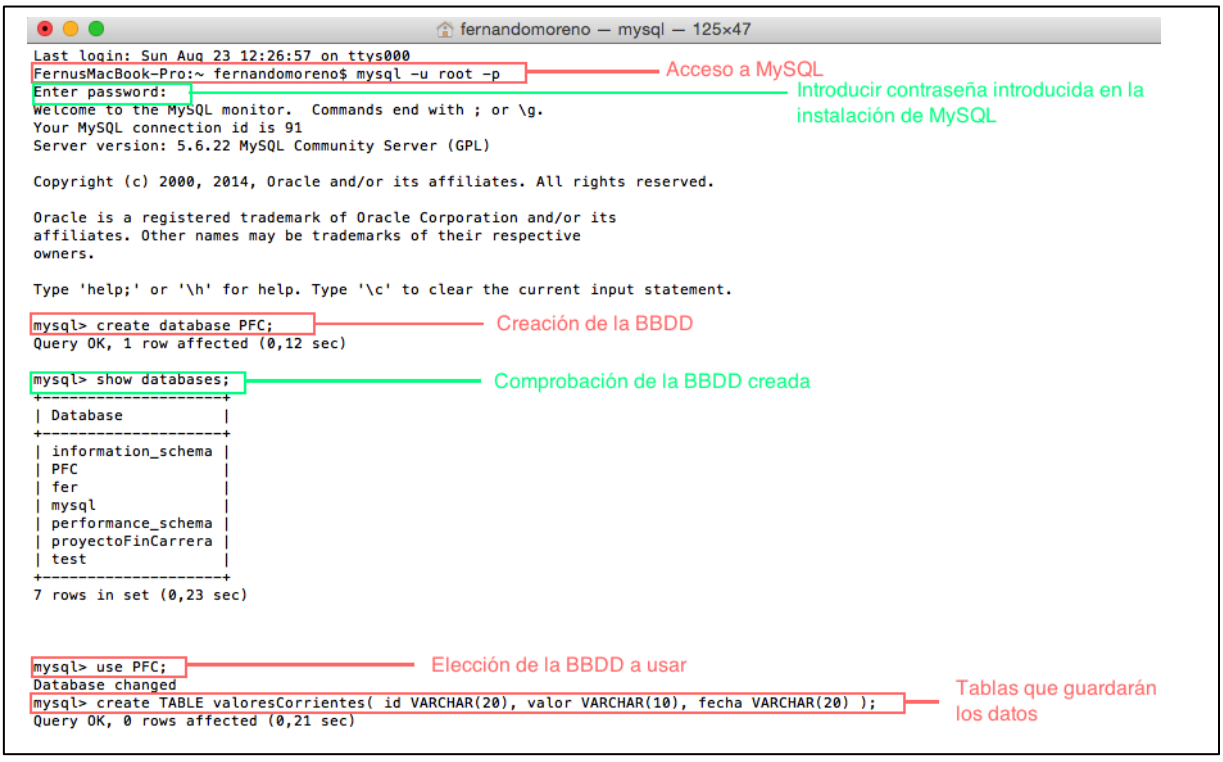

Figura 68: Creación de la BBDD en el servidor

5. Además para poder comunicar el código del proyecto con la BBDD es necesario descargarse un driver o conector, disponible en:

   http://dev.mysql.com/downloads/connector/j/5.0.html

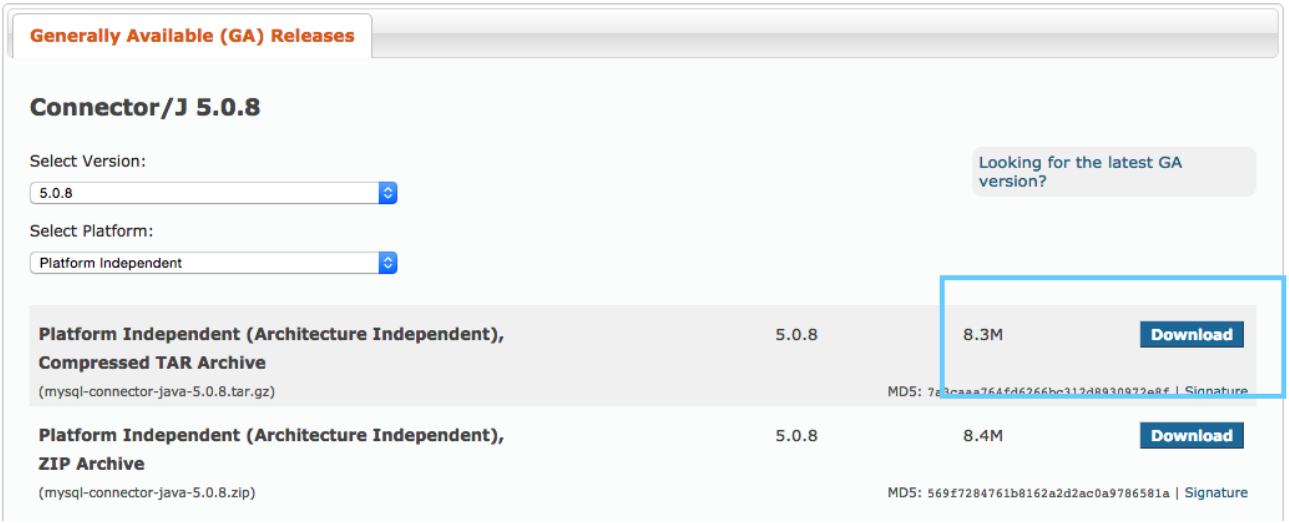

Figura 69: Instalación del driver

6. A partir de este momento ya se puede ejecutar la aplicación java, PFC_main.java.

Aplicación Android

A continuación se mostrarán los pasos necesarios para poder poner en ejecución la aplicación.

A. Instalación de Android Studio

El primer paso consiste en descargar en el ordenador la aplicación Android Studio disponible en https://developer.android.com/sdk/index.html

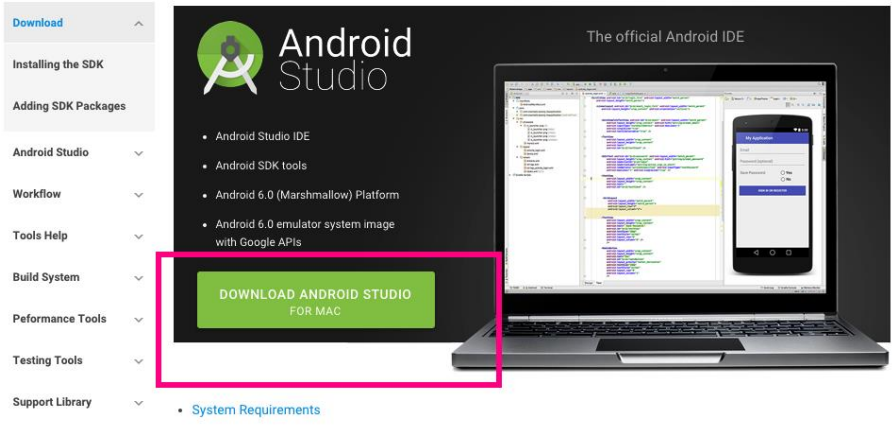

Figura 70: Instalación de Android Studio

B. Descarga del código

El siguiente paso consiste en descargar el proyecto Android y abrirlo en la aplicación anteriormente descargada.

A continuación se deberá seleccionar la dirección IP del servidor así como la del puerto utilizado. Para ello hay que editar el archivo PFC.java y en el método SendMessage() escribir la dirección IP que tiene nuestro servidor y el puerto 4444:

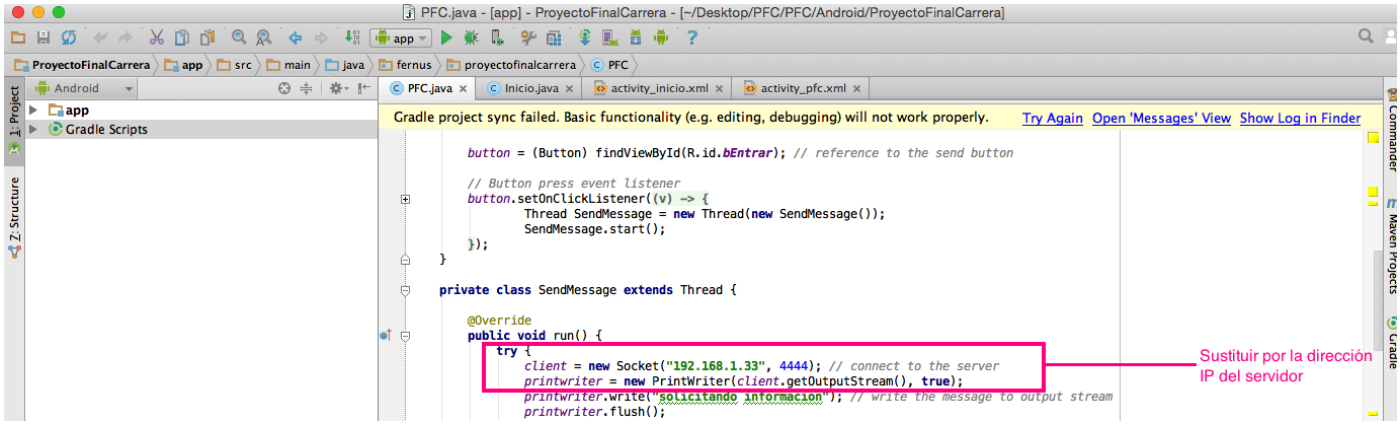

Figura 71: Selección de la dirección IP el servidor en dispositivo Android

El código se instalará en el dispositivo Android, conectándolo al ordenador mediante un cable USB y pulsando el botón run, resaltado en la siguiente imagen.

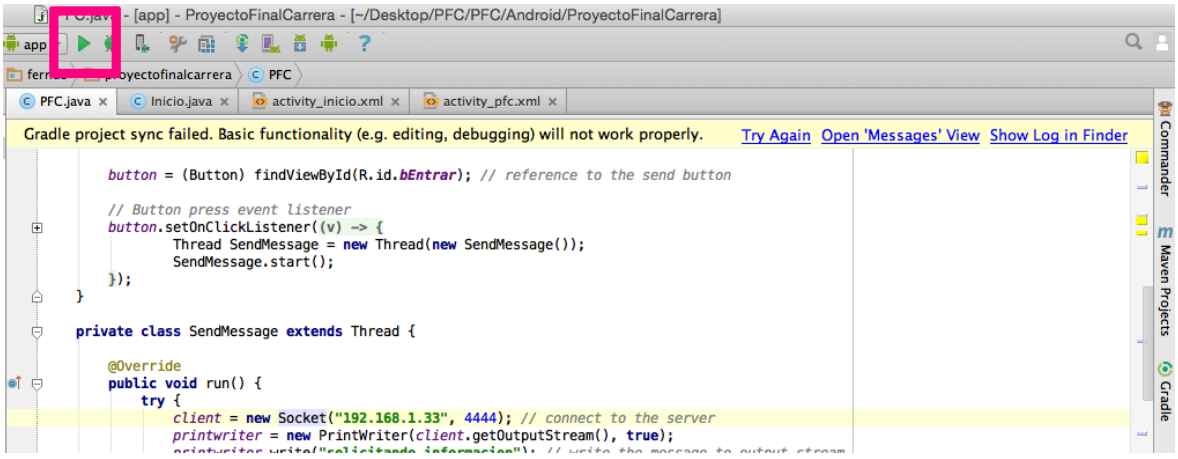

Figura 72: Botón run en la aplicación Android Studio

Una vez se han hecho todos los pasos anteriores, la aplicación estará lista para utilizarse

Anexo II: Hardware de la solución

En esta sección se muestran las características técnicas con las que cuenta la Waspmote y la de todos los módulos que se conectarán a ésta para llevar a cabo la realización de este proyecto [11].

A. Waspmote

Waspmote se basa en una arquitectura modular. El usuario puede cambiar o integrarlos módulos necesarios en los diferentes sockets según las necesidades.

Los módulos disponibles para integrar en Waspmote se clasifican en:

- ZigBee/802.15.4 / Módulos XBee (2.4GHz, 868MHz, 900MHz).
- Módulo LoRa (868/900MHz)
- Módulo GSM/GPRS (Quad-band: 850MHz/900MHz/1800MHz/1900MHz)
- Módulo 3G/GPRS (Tri-Band UMTS 2100/1900/900MHz and Quad-Band GSM/EDGE, 850/900/1800/1900 MHz)
- Módulos WiFi
- Módulos Bluetooth: Bluetooth Low Energy and Bluetooth Pro
- Módulo GPS
- Módulos NFC/RFID
- Módulos con sensores (corriente, humedad, temperatura, etc…)
- Módulo de almacenamiento: SD Memory Card

En las siguientes imágenes se muestra la placa Waspmote con dos ángulos de visión distintos, una vista superior y una vista inferior.

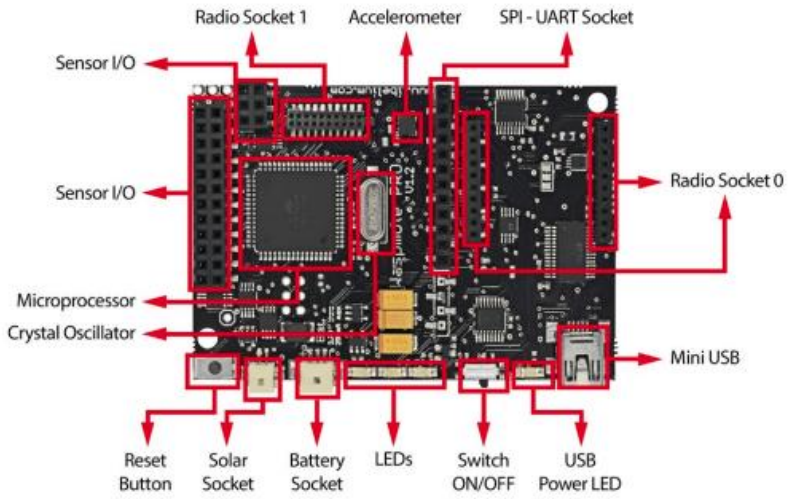

Figura 73: Vista superior de la Waspmote

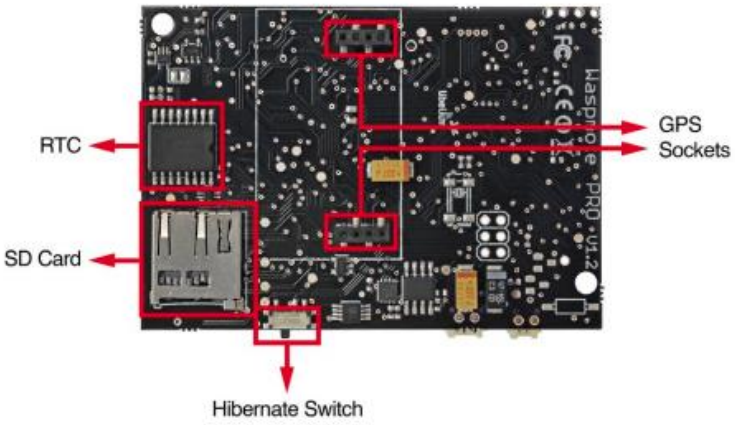

Figura 74: Vista inferior de la Waspmote

A continuación, se muestran las principales características hardware.

Datos generales

Microcontrolador	ATmega1281
Frecuencia	14.7456 MHz
SRAM	8KB
EEPROM	4KB
FLASH	128KB
Tarjeta SD	2GB
Peso	20gr
Dimensiones	73.5 x 51 x 13 mm
Rango de temperaturas	[-10ºC, +65ºC]
Reloj	RTC (32KHz)

Tabla 3: Datos técnicos generales de la Waspmote

Características eléctricas

Voltaje de la batería	3.3V – 4.2V
Carga USB	5V – 100mA
Carga por paneles solares	6 – 12V – 280mA

Tabla 4: Características eléctricas de la Waspmote

Diagrama de bloques

En la figura se pueden observar los diferentes bloques que componen el Waspmote y las señales de datos.

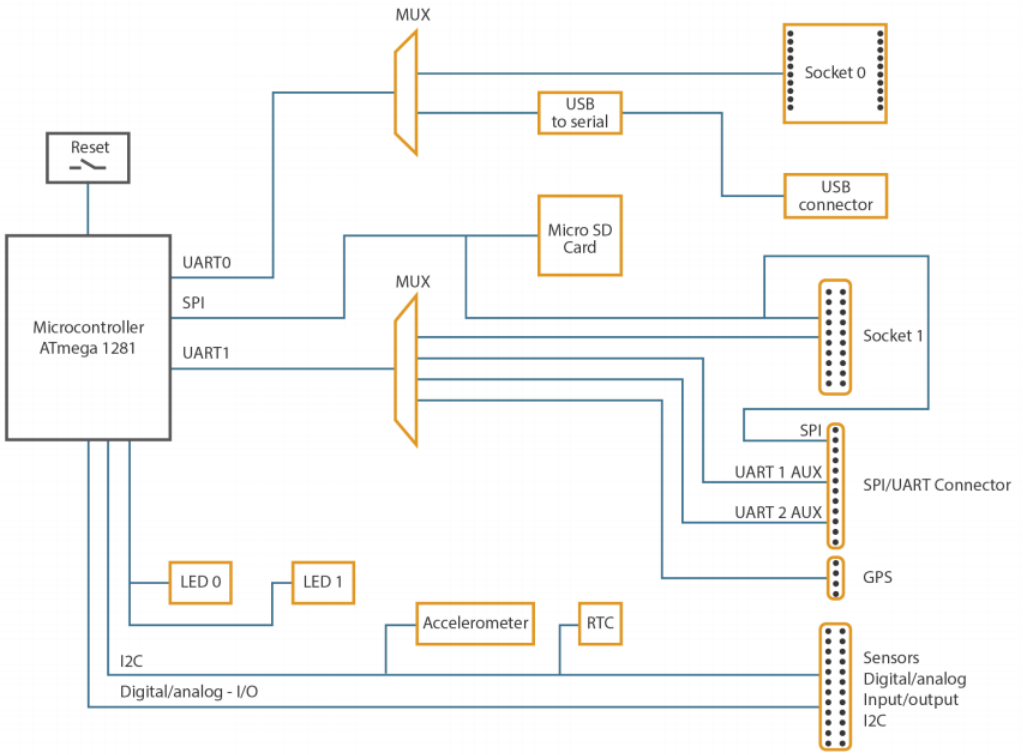

Figura 75: Diagrama de bloques de la Waspmote – Señales de datos

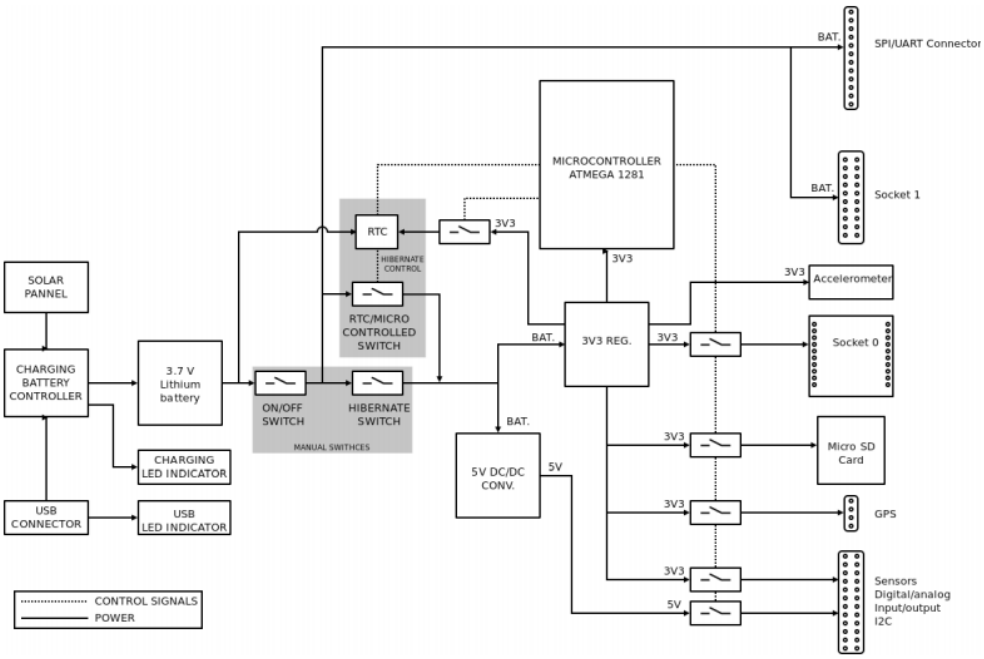

Figura 76: Diagrama de bloques de la Waspmote – Señales de potencia

Entradas/Salidas:

Waspmote puede comunicarse con otros dispositivos externos mediante los diferentes puertos de entrada/salida que posee.

Los dispositivos con los que la Waspmote se puede comunicar pueden ser cualquier sensor, componente o módulo electrónico siempre y cuando se respeten las especificaciones requeridas para cada puerto.

Dispone de dos conectores, el "Sensor I/O" y el "I2C UART socket" como se puede observar en la anterior figura.

Las entradas y salidas del Waspmote se pueden resumir en las siguientes:

- **Analógicas**. 7 entradas accesibles en el conector de sensores y conectadas directamente al microcontrolador (éste tiene un ADC de 10 bits cuyo valor máximo de entrada es 3.3 V).

- **Digitales**. 8 pines digitales configurables como entrada o salida, cuyos valores lógicos 0 y 1 corresponden a 0V y 3.3V, respectivamente.

- **PWM** (Pulse Width Modulation). Existe un pin que permite simular una onda analógica entre 0 y 3.3 V mediante modulación por anchura de pulso.

- **UART**. Waspmote dispone de 6 puertos serie. Una de las UART del microcontrolador está conectada simultáneamente al módulo de comunicación XBee y al puerto USB (por tanto, no se pueden usar simultáneamente el XBee y el puerto USB).

  La otra UART del microcontrolador está conectada a un multiplexor de cuatro canales, pudiendo seleccionar desde el código cuál de las cuatro nuevas UART queremos conectar a la UART del microcontrolador. Estas cuatro nuevas UART están conectadas dela siguiente manera. Una está conectada a la placa GPRS, otra al GPS y las otras dos quedan accesibles al usuario en el conector I2C – UART auxiliares.

- **I2C**. En Waspmote también se utiliza el bus de comunicación I2C, donde se conectan en paralelo tres dispositivos: el acelerómetro, el RTC y el potenciómetro que configura el nivel de umbral de alarma por batería baja.
  En todos los casos el microcontrolador actúa como maestro (master) mientras que el resto de los dispositivos conectados al bus actúan como esclavos (slave).

- **SPI**. El puerto SPI del microcontrolador se utiliza para la comunicación de éste con la tarjeta micro SD. Todas las operaciones de uso del bus son realizadas por la librería específica de forma transparente.

- **USB**. La comunicación USB se utiliza en Waspmote para la comunicación con un ordenador. Esta comunicación permite la carga del programa al

microcontrolador y la comunicación de datos durante la ejecución del programa. Para la comunicación USB se utiliza una de las UART del microcontrolador y de la conversión al estándar USB se encarga el FT232RL. De esta forma en el ordenador tendremos un nuevo puerto de comunicación serie (virtual) listo para comunicarse con Waspmote.

B. Sensor de corriente

El sensor de corriente es el encargado de medir la corriente que consume el dispositivo eléctrico que se quiere medir. Para ello es preciso conectarlo al socket1 de una placa adicional que debe ser conectada en la parte superior de la Waspmote

Se trata de un sensor de bajo coste perteneciente al bloque "Smart metering" de Libelium que genera una corriente proporcional a la corriente en el circuito principal. Esta corriente se convierte a voltaje mediante un resistencia de carga, obteniéndose una señal que la Waspmote pueda entender (conversor analógico - digital).

Figura 77: Sensor de corriente

Las características técnicas del sensor que se va a utilizar son las siguientes:

Corriente primaria máxima	100A
Turns ratio	1:2000 aproximadamente
Mínima resolución	130mA aproximadamente
Rango de medida	500mA – 40A

Tabla 5: Datos técnicos del sensor de corriente

Para usar dicho sensor, éste debe de engancharse al cable de alimentación del dispositivo para poder medir el consumo. A continuación se muestra gráficamente como debe de realizarse dicha conexión:

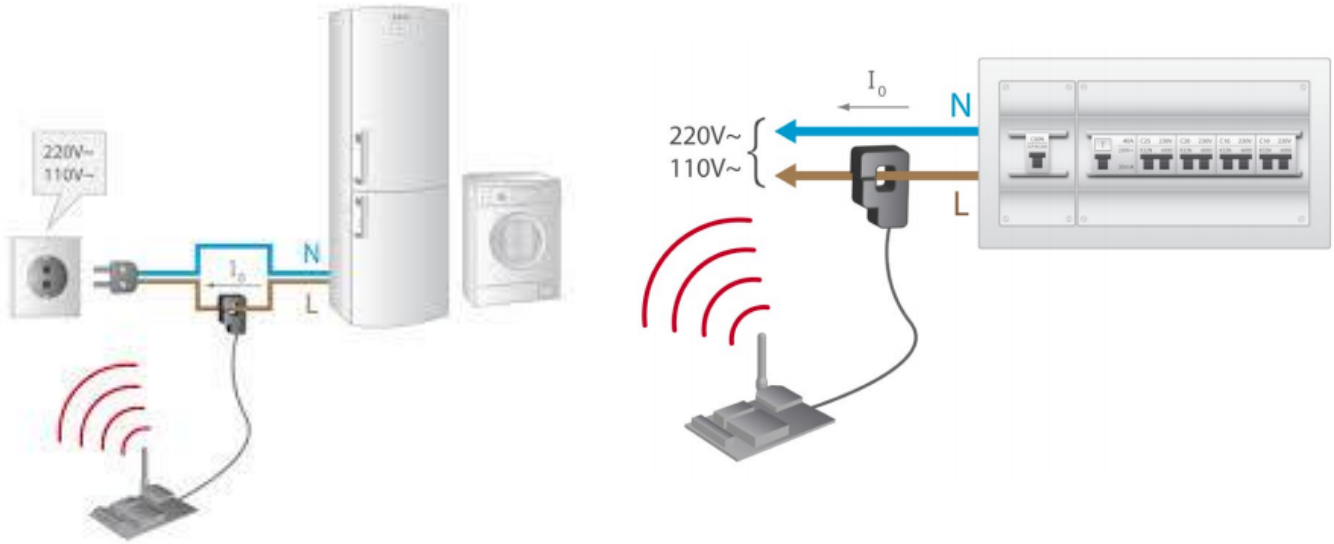

Figura 78: Distintas maneras de conectar el sensor de corriente

Respecto al modo de enganchar el sensor, hay que resaltar dos formas de realizarlo.

La primera singularidad a destacar, como se puede observar en la anterior figura, es que puede haber dos maneras de enchufar el sensor de corriente. La primera trata de enganchar el sensor a un sólo dispositivo eléctrico, como puede ser el caso de un frigorífico o una lavadora, etc.… para así poder llevar un control del consumo sobre un dispositivo concreto. Y la segunda opción es si se quiere monitorizar el consumo total de un hogar, conectando el sensor sobre el cuadro eléctrico (medidor de consumo total).

Señalar que el sensor de corriente se debe conectar al cable neutro, y no al cable de fase o al cable entero. A este fin se ha utilizado un cable de alimentación preparado para este fin como muestra la siguiente figura

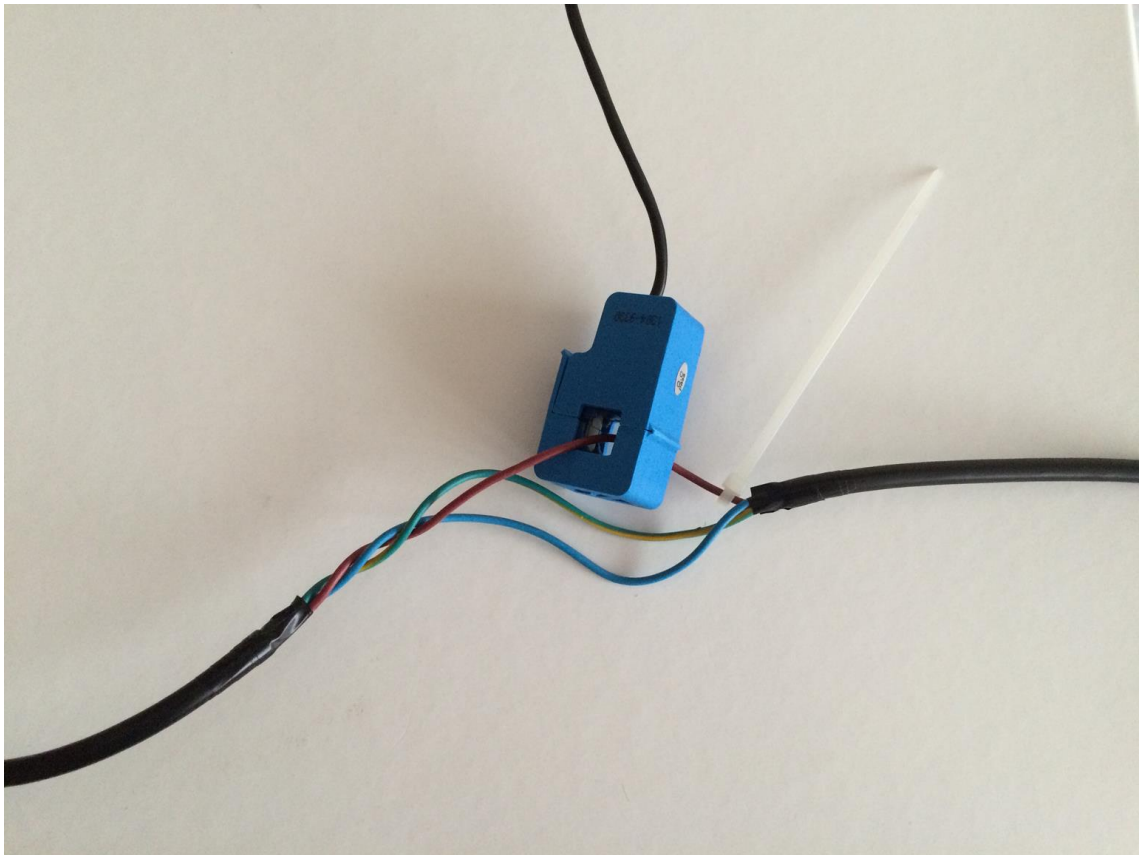

Figura 79: Modo de conectar el sensor de coriente

C. Antena WiFi

La antena WiFi es necesaria para el envío bidireccional de información entre la Waspmote y la aplicación cliente localizada en el ordenador.

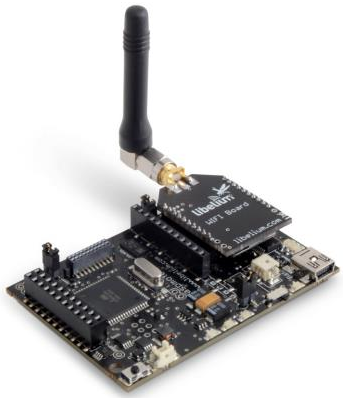

Figura 80: Antena WiFi

Las principales características técnicas de esta antena son las siguientes:

- **Protocolo**: 802.11b/g - 2.4GHz

- **Potencia de transmisión**: 0dBm - 12dBm (variable por software)
- **Sensibilidad del Receptor**: -83dBm
- **Conector de la antena**: RPSMA
- **Ganancia de la antena**: 2dBi (existe un modelo de 5dBi)
- **Seguridad**: WEP, WPA, WPA2
- **Topología**: AP(Access point)
- **Capacidad de roaming 802.11**
- **Distancia:** 50- 100m

Sistema energético.

La Waspmote tiene 4 modos de funcionamiento:

- **ON**: modo normal de funcionamiento. El consumo en este estado es de 9mA.

- **Sleep**: El programa principal se detiene, el microcontrolador pasa a un estado de latencia, del que puede ser despertado por todas las interrupciones asíncronas y por la interrupción síncrona generada por el Watchdog. El intervalo de duración de este estado va de 32ms a 8s. El consumo en este estado es de 62µA.

- **Deep Sleep**: El programa principal se detiene, el microcontrolador pasa a un estado de latencia del que puede ser despertado por todas las interrupciones asíncronas y por la interrupción síncrona lanzada por el RTC. El intervalo de este ciclo puede ir de 8 segundos a minutos, horas, días. El consumo en este estado es de 62µA.

- **Hibernate**: El programa principal se detiene, el microcontrolador y todos los módulos de Waspmote quedan completamente desconectados. La única forma de volver a activar el dispositivo es a través de la alarma previamente programada en el RTC (interrupción síncrona). El intervalo de este ciclo puede ir de 8 segundos a minutos, horas, días. Al quedar el dispositivo totalmente desconectado de la batería principal el RTC es alimentado a través de una batería auxiliar de la que consume 0,7µA.

Las acciones que pueden ser realizadas por dicha antena son las siguientes:

- Conexiones usando sockets TCP/IP - UDP/IP
- Conexión web HTTP
- Transferencia de archivos FTP
- Conexión con cualquier estándar router WiFi
- Asignación de direcciones IP automáticamente mediante DHCP

Para la realización de las comunicaciones se optado por hacerlo dentro de una red WiFi domestica con la ayuda de un router, aunque podría ser llevado a un ámbito más extenso.

D. RTC

El RTC (Real Time Clock) o reloj de tiempo real integrado de tipo DS3231SN de Maxim, mantiene informado al Waspmote del momento temporal en el que se encuentra. Se programa a través del bus I2C y permite programar acciones, como por ejemplo, hacer que el mote hiberne y se despierte en un determinado momento gracias al uso de interrupciones.

Se dispone de un reloj en tiempo real a 32KHz (32.768Hz) que permite establecer una base de tiempos absoluta para la utilización del dispositivo.

E Batería

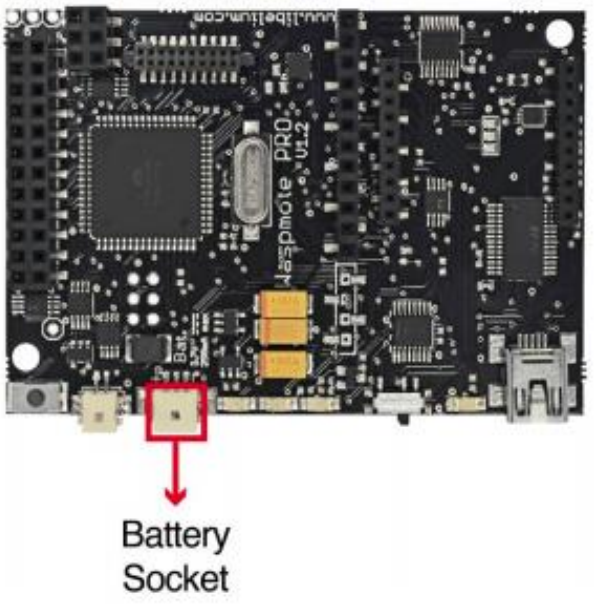

Figura 81: Socket de la bateria

Para alimentar el Waspmote hay 3 opciones:

- Batería: Se incluye una batería de Ion de Litio de tensión nominal 3.7V. En lo referente a la capacidad de la batería existen dos posibilidades: 1150 mAh y 2300 mAh.

- Panel solar: Waspmote dispone de un conector y un cable especial para conectar a un panel solar. Se admiten placas solares de hasta 12V y corriente máxima de carga a través de placa de 240 mA.

- USB: Permite conectar mediante un cable USB a un PC, una fuente de 220V o un conector de vehículo.

Características eléctricas

Voltaje de la batería	3.3V – 4.2V
Carga USB	5V – 100mA
Carga por paneles solares	6 – 12V – 280mA

Tabla 6: Características eléctricas de la batería

Se ha elegido la opción de la batería para poder dar un uso libre y portátil al proyecto

F. Memoria SD

Figura 82: Socket tarjeta SD

Waspmote integra un soporte de almacenamiento externo como son las tarjetas micro-SD (Secure Digital). Waspmote usa el sistema de ficheros FAT16 y puede soportar tarjetas de hasta 2GB y la comunicación desde el microcontrolador hasta la tarjeta se realiza mediante el bus SPI. Esta es una alternativa para grabar datos de manera masiva frente a la EEPROM.

Anexo III: Presupuesto

En este apartado se muestra el gasto incurrido en la realización de este proyecto, incluyendo el gasto de personal para el desarrollo del sistema y los equipos utilizados para ello.

Primeramente se enumerarán los materiales necesarios para llevar a cabo la toma de las muestras de corriente [19]:

Componentes	Precio	Precio tras amortización
Waspmote + módulo WiFi 2dBi	158€	21,06
Sensor de corriente (0-100A)	20€	2,66
Smart Metering sensor board[1]	95€	12,67
Batería	18€	2,4
Micro SD 2GB	6€	0.8
Coste total	297€	29,59 €

Tabla 7: Coste componentes Waspmote

Para realizar la amortización se ha usado la siguiente fórmula dada por la universidad:

$$\frac{A}{B} xCxD$$

En donde:

A = número de meses desde la fecha de facturación en que el equipo es utilizado
B = periodo de depreciación (60 meses)
C = coste del equipo (sin IVA)
D = % del uso que se dedica al proyecto (habitualmente 100%)

Además se tiene que considerar los siguientes componentes para hacer que el proyecto sea realizable.

Componentes	Precio	Precio tras amortización
MacBook Pro	1.449€	193,2
Samsun S4 mini	190€	25,33
Router	33€	4,4
Coste total	1.672€	222,93 €

Tabla 8: Coste componentes varios

[1] Placa necesaria para conectar el sensor de corriente encima de la Waspmote

Además se incluyen **gastos varios** en conectores, polímetro y cables por un valor de 150 €.

La duración del desarrollo completo del proyecto asciende a un total de 1500 horas de trabajo.

Recursos profesionales	**Horas invertidas**	**Precio/hora**	**Coste total**
Programador	1500	15 €	22.500

Tabla 9: Costes trabajo de programación

A continuación se va a mostrar un desglose del tiempo dedicado aproximado para cada una de las fases del proyecto:

Tareas	Día de comienzo	Día de finalización	Número de hora
Primeros pasos con Waspmote	01-02-2014	20-02-1014	60
Sensor de corriente	15-02-2014	10-03-2014	100
Módulo de memoria SD	28-02-2014	16-03-2014	100
Envío/recepción información al servidor	10-09-2014	20-11-2014	200
Total Waspmote	01-02-2014	20-11-2014	460 horas
Recepción/envió información a la Waspmote	01-09-2014	15-11-2014	100
Instalación MYSQL y creación tablas en la BBDD	28-10-2014	15-11-2014	100
Conexión BBDD con el servidor	10-11-2014	20-11-2014	50
Creación del Menú en el servidor	21-11-2014	15-01-2015	150
Conexión con la aplicación Android	01-12-2014	10-01-2015	120
Total Servidor	01-09-2014	10-01-2015	570 horas
Primeros pasos en Android	20-12-2015	10-01-2015	120
Creación del icono de la App y pantallas	07-12-2015	10-12-2015	50
Envío/recepción de información al servidor	25-12-2014	28-01-2015	150
Total App Android	20-12-2014	28-01-2015	320 horas
Documentación proyecto	01-06-2015	01-09-2015	150 horas
Total	04-02-2015	01-09-2015	1500 horas

Tabla 10: Planificación temporal del proyecto

El coste total para la realización de este proyecto es:

Concepto	Precio
Waspmote + módulos	29,59 €
Componentes	223,93 €
Costes de personal	22.500 €
Costes varios	150 €
Total	22.903,52 €

Tabla 11: Presupuesto total del proyecto

El presupuesto total de este proyecto asciende a la cantidad de 22.903,52 Euros.